REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-04-
0040

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	31 August 2003	Final Performance Report

4. TITLE AND SUBTITLE

Optimal Scheduling with Flexible Resources

5. FUNDING NUMBERS

C — F49620-01-1-0222

6. AUTHOR(S)

Leyuan Shi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Industrial Engineering
University of Wisconsin-Madison
1513 University Ave.
Madison, WI 53706

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research
Optimization and Discrete Mathematics
Program Manager: Juan Vasquez
4015 Wilson Blvd., Room 824A
Arlington, VA 22203-1954

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

In this project, the PI has developed an efficient optimization framework, the Nested Partitions (NP) method, for difficult combinatorial optimization problems (see publications). In particular, the Job Shop Total Inventory Minimization Problem (JSTIMP) and the Multicommodity Distribution System Design (MDSD) Problem were investigated. It has been demonstrated that the NP framework can be integrated with many well-known optimization algorithms/methods to provide hybrid algorithms for efficiently solving many large-scale combinatorial optimization problems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Combinatorial Optimization, Planning, Scheduling, Nested Partitions, Flexible Resources	74
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

OPTIMAL SCHEDULING WITH FLEXIBLE RESOURCES

Final Performance Report

August 31, 2003

Grant Number: F49620-01-1-0222

PI: Leyuan Shi
Department of Industrial Engineering
University of Wisconsin
1513 University Avenue
Madison, Wisconsin 53706

Tel: (608) 265-5969
Fax: (608) 262-8454
Email: leyuan@engr.wisc.edu

Abstract

In this project, the PI has developed an efficient optimization framework, the Nested Partitions (NP) method, for difficult combinatorial optimization problems (see publications). In particular, the Job Shop Total Inventory Minimization Problem (JSTIMP) and the Multicommodity Distribution System Design (MDSD) Problem were investigated. It has been demonstrated that the NP framework can be integrated with many well-known optimization algorithms/methods to provide hybrid algorithms for efficiently solving many large-scale combinatorial optimization problems.

Contents

1. Executive Summary

Research under this grant emphasized the development of theory and efficient implementations of hybrid NP algorithms for various classes of large-scale combinatorial optimization problems. With these methods we have been able to solve problems that are intractable for other approaches. In particular, a major area of investigation under this grant has been the solution of the Job-Shop Total Inventory Minimization problem (JSTIMP) and the Multicommodity Distribution System Design (MDSD) Problem. We have focused on the complexity issue of the JSTIMP and developed theoretical results (Pan 2003, Chapter 4). For MDSD problem, we developed a process that is implemented via looping constructs available in the AMPL modeling language (with CPLEX being used as the branch-and-cut solver), and thus represents a novel framework for the utilization of modeling-language/branch-and-cut software for large-scale combinatorial optimization problem. As a significant new addition to the arsenal of optimization methodologies for tackling problems of practical interest, NP and hybrid NP methods are worth further investigation in order to leverage their potential in other application domains.

2. Introduction

2.1 Background

Planning and scheduling are some of the most important functions within many production systems. The objective of planning and scheduling is to maximize an appropriate measure of productivity by ensuring the efficient use of raw materials and various resources, including labor and machines, involved in the manufacturing process. Today's globally competitive economy further strengthens the traditional role played by planning and scheduling.

In a competitive environment, effective planning and scheduling have become a necessary condition for survival in the marketplace. As a direct consequence of intensified competition from business rivals overseas, domestic firms can no long afford the "luxury" of certain inefficiencies in their processes that were previously considered standard industry practices. This means that manufacturing firms and their partners in academia need to work hard together to find solutions.

Another complicating factor that puts still more pressure on the planning and scheduling functions is the advent of flexible manufacturing resources. One example would be manufacturing cells that consist of Computer Numerically Controled (CNC) machines and industrial robots. One other example would be cross-trained workforce capable of undertaking a diverse set of tasks. This kind of resource flexibility opens doors to new possibilities of greater productivity. However, it also poses significant new challenges, which entail much more beyond the theories and methods provided by the classical planning and scheduling research. Our research is aimed at bridging this gap.

With recent advancements in computing power and optimization methodologies, the development of advanced planning and scheduling tools has almost come with reach. Such tools must be able to communicate, collaborate, and integrate their planning and scheduling functionalities to obtain optimal results throughout the manufacturing enterprise. This research project is intended to bring state-of-the-art decision and optimization methodologies to bear in meeting this need.

2.2 Objectives

Specifical objectives of this research are to

- contribute to the state-of-the-art in scheduling theory by expanding its traditional scope into the area of flexible resource scheduling,
- provide new optimal and heuristic scheduling algorithms for simultaneously allocating flexible resources and scheduling jobs,
- generate tools to improve daily scheduling decisions in manufacturing systems that already have flexible resources,
- develop new methodology for determining the level of cross-training needed and desired in a manufacturing firm,
- provide guidelines for which production environments can realize the full benefits of resource flexibility,
- contribute to the state-of-the-art in simulation-based optimization through new algorithms and stopping rules that can be applied for discrete stochastic optimization using simulation, and
- provide a test-bed of discrete stochastic optimization problems, and by demonstrating the feasibility of simulation-based optimization for real-life problems.

3. Methods, Assumptions, and Procedures

3.1 The Nested Partitions Method

The proposed methodology, the Nested Partitions (NP) method (Shi and Ólafsson 2000), stems from a combination of recent theoretical and technological advances which can explicitly tackle the inherent complexity of many scheduling problems. The NP method, a newcomer on the stage of optimization metrologies, is a significant addition to classical Linear Programming (LP)-based, mixed integer solvers and heuristic-based search algorithms. Designed to take advantage of domain knowledge, NP is better suited for incorporating heuristics, such as tabu search (Glover, McMilan, and Novick 1989, Glover, McMilan, and Novick 1990) and genetic algorithms (Goldberg 1989), than LP-based solvers. NP reflects the realization that on one hand, problems of practical interest are often too difficult to solve to optimality, and that on the other hand, a "good," quick solution is all that is called for.

The NP method provides a powerful new optimization framework that combines adaptive global sampling with local heuristic searches or domain knowledge. It uses a flexible partitioning method to divide the search space into regions that can be analyzed individually and then aggregates the results from each region to determine how to continue the search, that is, where to concentrate the computational effort. This partitioning approach makes the NP framework uniquely well suited to distributed systems environments. Furthermore, with the NP framework there are multiple tasks that can be distributed and the degree of parallelism is very flexible. It is therefore

also well suited to environments with heterogeneous resources, where machines may alternate between being available or busy during the execution of the optimization algorithm, as may be the case for distributed web-based computing.

The NP method adaptively samples from the entire feasible region, or search space, and concentrates the sampling effort by systematically partitioning the feasible region. In each iteration of the algorithm, it maintains the most promising region, that is, the sub-region that is considered to be the most likely to contain the global optimum. This most promising region is then partitioned into a given number of sub-regions, these sub-regions and the surrounding region are sampled, and the sampling information is used to determine which region should be the most promising region for the next iteration. However, instead of using the sample points directly, a local search can be run with the sample point merely as a starting point, and the final result of the local search used to represent that region. To incorporate such local searches, the NP framework has been successfully combined with both general-purpose heuristics such as genetic algorithms (Shi, Ólafsson, and Chen 2001), tabu search (Shi and Men 2003, Chalermdamerichai 1999), and special purpose heuristics such as some well-known heuristics for the traveling salesman problem (Shi, Ólafsson, and Chen 1999). Thus, a defining feature of the NP framework is that it can incorporate any heuristic search or domain knowledge effectively. This is of great importance for it to be applicable as a generic optimizer for a large variety of problems. Indeed, it can be thought of as a meta-heuristic that guides the search of any general purpose or specialized optimization algorithm. Using the NP framework, the proposed optimization engine will be very flexible in incorporating algorithms such as tabu search, greedy search, and other state-of-the-art general purpose algorithms, as well as the option of incorporating user supplied specialized algorithms.

3.2 Modeling Assumptions

We have devoted most of our effort to problems where the input is deterministic, while some of our previous work focuses on stochastic problems (e.g., we studied the scheduling of the manufacturing cell environment in Ólafsson and Shi (2000)). Undoubtedly, real-life problems almost inevitably involves some probabilistic elements. However, a thorough understanding of a deterministic problem is a necessary step toward understanding its stochastic counterpart. Furthermore, the NP framework, which utilizes random sampling, simulation, and promising index estimation, provides a natural means of incorporating heuristics developed in the research, and extending the solution of a deterministic problem to its more realistic, and thus more useful, stochastic extension.

In those situations where stochastic problems are discussed, we have assumed exponential job processing times.

3.3 Procedures for Scheduling Flexible Resources

We introduce a new model that optimizes a job shop schedule with respect to a performance measure called the *total inventory*. This work was motivated by our joint project with John Deere Horicon Works (located in Horicon, Wisconsin) to help one of their suppliers to revamp its scheduling function. For the job shop total inventory problem, we introduce several single-machine relaxations and discuss their complexity. Job reentrancy is also considered. Two approximation algorithms are proposed, one based on an insertion technique and the other based on a shifting bottleneck

procedure. The effectiveness of the proposed algorithms is demonstrated through extensive computational experiments.

Additional efforts are made to address issues espoused by the total inventory minimization problem. Firstly, we further study the computational complexity of scheduling jobs on a machine to minimize the sum of completion times subject to delayed reentrancy or delayed precedence constraints. Job reentrancy occurs frequently in various manufacturing processes. Its delayed version, where a certain amount of delay is required between two consecutive visits by a job to the same machine, is very common, too. We study the complexity of delayed reentrancy in connection with delayed precedence constraints, which is an area that has stimulated a number of research efforts in recent years. Both preemptive and nonpreemptive schedules are considered.

Secondly, we consider a single-machine scheduling problem with release dates and deadlines to minimize the weighted sum of completion times. We propose the first nontrivial lower bound for this problem and define it as the optimal objective value of a particular linear program. The linear program is then reformulated as an optimization problem with a piecewise linear, concave objective function subject to inequality and non-negativity constraints. An $O(n \log n)$ procedure is developed to compute the lower bound, which is further tightened at an additional expense of $O(n^2 \log n)$ time. We then incorporate the lower bound into a branch-and-bound solution procedure. To curtail the search, we develop a node elimination technique that calls the solution procedure recursively. Extensive computational experience confirms the effectiveness of our method.

Thirdly, we consider a scheduling problem in which n jobs with distinct deadlines are to be scheduled on a single machine. The objective is to find a feasible job sequence that minimizes the sum of weighted completion times. We present an efficient branch-and-bound algorithm that fully exploits the principle of optimality. Favorable numerical results are also reported on an extensive set of problem instances of 20–120 jobs.

Finally, we investigate the single-machine scheduling problem that minimizes the maximum lateness. Despite its being strongly NP-hard (Garey and Johnson 1979), this problem has been thought to be computationally inexpensive to solve using Carlier's branch-and-bound algorithm. Carlier's algorithm (CA) has achieved excellent results on some randomly generated instances consisting of 50–10,000 jobs. However, we find that hard instances of this problem exist for even moderate size. We identify many such instances, each of which poses an insurmountable challenge to CA. We propose a new lower bound and a problem size reduction procedure, and consider both the problem and its inverse. Making use of the new techniques as well as incorporating a number of existing ones, we have designed and tested a new branch-and-bound algorithm that substantially outperforms CA in computation time. More importantly, the new algorithm is significantly less susceptible to hard instances.

Machine shops are placing strong emphasis on innovative technologies to gain competitive edge through superior quality, flexibility, and responsiveness. With two spindles and two independent controlled turrets, automated material handling, as well as mill/turn capability, four-axis CNC (Computer Numerically Controlled) turning centers allow turned components with high degree of complexity to be produced quickly on a single machine.

The improved processing capabilities of four-axis CNC turning centers have also led to increased complication in process planning for these machines. Manual generation and optimization of process plans is highly inefficient due to the combinatorial characteristics that arise from multiple spindles and turrets. We propose a Nested Partitions (NP) algorithm to the process plan optimization problem for four-axis turning centers. The experiments show that the NP algorithm

is capable of rapidly producing high quality process plan and thus, is well suited for the user-interactive computer-aided-process-planning framework in our context. Comparisons of results from tabu search heuristic and branch-and-bound algorithm are also conducted.

Our most recent work is concerned with applying the NP method to problems in domain of logistics and supply chain optimization. From a computational viewpoint, supply chain network design problems are quite challenging because they are combinatorially explosive—some of these optimization models have millions of variables and millions of constraints. Unsurprisingly, the problem belongs to the class of NP-hard problems (Garey and Johnson 1979), which means that any attempt to obtain an optimal solution for every instance of the problem will require unacceptable levels of computation time. Our research provides new global hybrid algorithms and software tools for large-scale supply chain optimization and extends the range of problems for which the hybrid randomized optimization approach significantly outperforms alternatives such as Branch & Cut, Lagrangian Relaxation and heuristic search techniques not based on partitioning. We tested our hybrid algorithm with different size test problems—the largest test problem having 15 products, 10 plants, 100 possible warehouse locations and 250 customers.

4. Results and Discussion

4.1 The Job Shop Total Inventory Minimization Problem

We consider scheduling problems in a job shop with m *machines* $\mathcal{M} = \{1, \ldots, m\}$ and n *jobs* $\mathcal{J} = \{1, \ldots, n\}$. Each job entails a chain of *operations* $\mathcal{O}_j = \langle f_j, \ldots, l_j \rangle$, where f_j and l_j are the first and the last operations, respectively. Each operation u can only be performed on a prescribed machine $m(u)$, taking $p(u)$ units of time. The machines can perform at most one operation at a time. Let binary relation $\mathcal{A}_j$ on $\mathcal{O}_j$ ($j \in \mathcal{J}$) represent the precedences between the operations of job j, and binary relation $\mathcal{E}_k$ represent the pairs of distinct operations performed on machine $k \in \mathcal{M}$. The former is *transitive* because u preceding v and v preceding w imply u preceding w. The latter is *symmetric* because if u and v are performed on the same machine, so are v and u. Notions from set theory (Mendelson 1997) can facilitate exhibition. A *partial order* is a binary relation $\mathcal{R}$ on a set $\mathcal{F}$ such that $\mathcal{R}$ is transitive and $\forall x \in \mathcal{F}, \langle x, x \rangle \notin R$. $\langle x, y \rangle \in \mathcal{R}$ and $\langle y, x \rangle \in \mathcal{R}$ cannot simultaneously hold. A *total order* is a partial order such that $\forall x \in \mathcal{F}, y \in \mathcal{F}, x \neq y$, either $\langle x, y \rangle \in \mathcal{R}$ or $\langle y, x \rangle \in \mathcal{R}$. In the current context, $\mathcal{A}_j$ is a total order on $\mathcal{O}_j$, and $\mathcal{A} := \cup_{j \in \mathcal{J}} \mathcal{A}_j$ a partial order on $\mathcal{O} := \cup_{j \in \mathcal{J}} \mathcal{O}_j$.

Let $S(u)$ and $C(u)$ denote the start time and the completion time of operation u. We are interested in finding nonpreemptive schedules, where $C(u) = S(u) + p(u)$. There are two reasonable requirements for any feasible schedule: (i) The operation precedences and (ii) the machine capacity constraints must be satisfied (Balas 1979). With our notation,

$$C(u) \leq S(v), \ \forall \langle u, v \rangle \in \mathcal{A}; \tag{1}$$

$$C(u) \leq S(v) \text{ or } C(v) \leq S(u), \ \forall k \in \mathcal{M}, \langle u, v \rangle \in \mathcal{E}_k. \tag{2}$$

Let $S_j := S(f_j)$ and $C_j := C(l_j)$ denote the start and the completion times of job j. CJSSP assumes that $S_j \geq 0$ for all j, and seeks for a schedule that minimizes the *makespan*, i.e., the maximum completion time of operations (denoted by $C_{\max} := \max_j C_j$).

Let $\bar{d}_j$ denote the deadline of job j. The underlying assumption is that there is sufficient capacity to handle the workload with the given job deadlines. This assumption is almost always satisfied

in shops with low-to-medium workload. In the cases where this assumption is violated, deadlines can almost always be extended through negotiations between the manufacturer and the customer. Assuming that deadlines will be met, our objective is to minimize the total cost due to WIP and FGI.

Let $D_j := \bar{d}_j - C_j$ denote the time elapsed between the completion of job j and the deadline. We define $Z_j^F := \mu_j^F D_j$ as a measure of the inventory holding cost incurred by the finished parts associated with job j, where μ_j^F is the FGI holding cost per unit of time. We define $Z_j^W := \mu_j^W (C_j - S_j)$ as a measure of the inventory holding cost attributed to the WIP associated with job j, where μ_j^W is the WIP holding cost per unit of time. In real life, it is almost always the case that $\mu_j^W \leq \mu_j^F$, which we assume here. Now we define the *total inventory cost* $Z := \sum_j Z_j^W + Z_j^F$ and give the following formulation of the problem

$$\begin{array}{lll} & \min & Z \\ \textbf{(JSTIMP)} & \text{s.t.} & (1), (2); \\ & & C_j \leq \bar{d}_j, \ \forall j \in \mathcal{J}. \end{array}$$

When $\mu_j := \mu_j^W = \mu_j^F$, the objective function is $Z = \sum_j \mu_j (\bar{d}_j - S_j)$. If we view **JSTIMP** from a perspective that is backward in time, we will find that deadlines become release dates (denoted by r_j), and that start times become completion times. If we properly define new jobs by reversing the chains of operations, the problem turns into one that minimizes the *sum of weighted flowtimes* (Morton and Pentico).

To the best of our knowledge, **JSTIMP** has not been addressed in the literature. The closest existing formulation is that of Ahmadi and Bagchi (1992) who consider a particular permutation flow shop problem in which the objective is equivalent to ours when $\mu_j^W = \mu_j^F = 1$ for all j; they propose an enumerative procedure for the case when $|\mathcal{M}| = 2$. Another related but more restrictive version is the single-machine weighted job earliness problem subject to no tardy jobs (without considering job release dates, however), for which Ahmadi and Bagchi (1986b) and Chand and Schneeberger (1986) investigate enumerative solution procedures. Ahmadi and Bagchi (1986a) also study the special case when all jobs have equal weights.

To simplify our analysis, we did not include any constraint that makes references to the left edge of the planning window. For example, we do not consider job ready times and machine ready times. This simplifying treatment is similar to that adopted by Ahmadi and Bagchi (1992) in studying their permutation flow shop problem. This guarantees that feasible schedules always exist for **JSTIMP**.

Modified disjunctive graph representation

The disjunctive graph of Roy and Sussmann (1964) can be used to visualize **JSTIMP** but with some modification. Take for an example a problem instance as shown in Figure 1, which illustrates the job processing times and routings, and the disjunctive arcs indicate the grouping of operations by machine. Moreover, we define job deadlines $\bar{d}_1 = 15$, $\bar{d}_2 = 20$, and $\bar{d}_3 = 17$. To bring the deadlines into the picture, we associate a dummy node o_∞^j with each job j, and add a sink node denoted by o_t. In addition, o_∞^j has a weight equal to $\bar{d}_{\max} - \bar{d}_j$, where $\bar{d}_{\max} = \max_j \bar{d}_j$, and o_t has zero weight. For the purpose of distinguishing between the original nodes and the auxiliary nodes, we refer to the former as *regular* nodes. Figure 2 details the extension that we have made to the

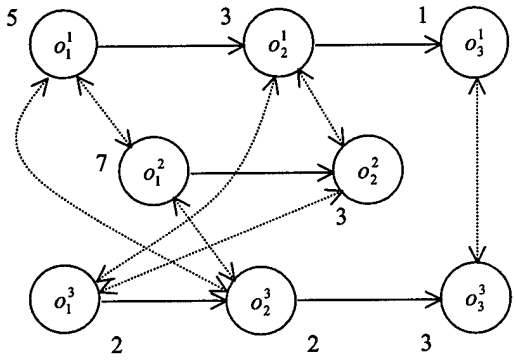

Figure 1: Disjunctive Graph

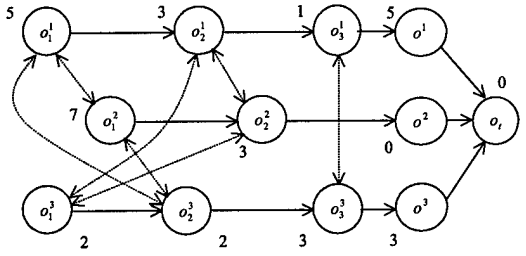

Figure 2: Modified Disjunctive Graph

basic disjunctive graph, as well as additional conjunctive arcs needed to connect regular nodes and auxiliary nodes. Figure 3 shows what the graph might look like after all disjuncts are resolved.

In the following, we consider the case where $\mu_j^W = \mu_j^F = \mu_j$ for all $j \in \mathcal{J}$. Let $\rho(u, v; \mathcal{G})$ denote the longest path length from u to v in a graph $\mathcal{G}$. Then, given an orientation $\mathcal{R}$, the total inventory Z is equal to $\sum_j \mu_j \rho(f_j, o_t; \mathcal{G})$, where $\mathcal{G} = (\mathcal{O}, \mathcal{A} \cup \mathcal{R})$. Hence, **JSTIMP** can be reformulated as

$$\begin{aligned} \min \quad & \sum_j \mu_j \rho(f_j, o_t; \mathcal{G}), \qquad (3) \\ \text{s.t.} \quad & \mathcal{G} = (\mathcal{O}, \mathcal{A} \cup \mathcal{R}), \mathcal{R} \text{ is a feasible orientation.} \end{aligned}$$

Solution Approaches

Due to the intractability of **JSTIMP**, we resort to approximation algorithms to solve it. Approximation algorithms are extensively employed in the context of machine scheduling. In this section, two efficient algorithms are proposed.

An insertion algorithm

The insertion algorithm is originally intended for the flow-shop scheduling problem (Nawaz et al. 1983). Werner and Winkler (1995) have later adapted it to solve CJSSP. The insertion algorithm

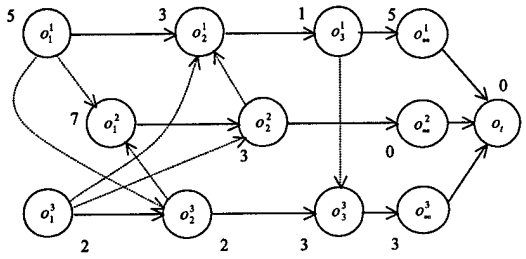

Figure 3: Graph Representing a Solution

can also be adapted for solving **JSTIMP**, the total inventory minimization problem.

Here is an outline of the insertion algorithm. Initially, no operations are sequenced on the machines, corresponding to graph $(\mathcal{O}, \mathcal{A})$. Jobs are sorted by some rule, taking into account their processing times and deadlines. Operations are then scheduled for each of the successive jobs in the list. Specifically, assume operation u is to be scheduled on its required machine $m(u)$ during an iteration step. Suppose that there are already l operations $\langle o_1, \ldots, o_l \rangle$ whose processing order has been determined on machine $m(u)$ (o_1 processed first, and o_l last). In addition, assume that $\mathcal{A}_0$ is the partial order on $\mathcal{O}$ that is the union of $\mathcal{A}$ and the precedences among the operations already scheduled on their required machines. We would like to optimally insert u among the other operations. Regardless of where we insert u, graph $(\mathcal{O}, \mathcal{A}_0)$ will be augmented with additional arcs, reflecting the new precedences (denoted by $\mathcal{R}(u, o_i)$) between u and $o_i, i = 1, \ldots, l$. It can be shown that there exists at least one position among the $l+1$ choices such that $\mathcal{A}_0 \cup (\cup_{i=1}^{l} \mathcal{R}(u, o_i))$ is a partial order on $\mathcal{O}$. Furthermore, we would like to choose the one that yields the minimal value with respect to some function of graph $(\mathcal{O}, \mathcal{A}_0 \cup (\cup_{i=1}^{l} \mathcal{R}(u, o_i)))$. Then, set $\mathcal{A}_0$ to $\mathcal{A}_0 \cup (\cup_{i=1}^{l} \mathcal{R}(u, o_i))$. When all operations are thus inserted, the algorithm will terminate with a solution represented by $\mathcal{A}_0$. A generic insertion algorithm is described below:

Generic Insertion Algorithm

Step 1. Sort jobs by rule R1.

Step 2. Starting with the first job in the sorted list, repeat the following until all operations of the current job are scheduled:

(a) Select an unscheduled operation of the current job by rule R2. Label it u.

(b) Suppose that there are l operations scheduled on machine $m(u)$. Among the $l+1$ potential choices, insert u in the position that yields the minimal value with respect to some function of graph $(\mathcal{O}, \mathcal{A}_0 \cup (\cup_{i=1}^{l} \mathcal{R}(u, o_i)))$ (denoted by R3). If two or more positions are equally preferable with respect to R3, choose among these positions with a tie-breaking rule R4.

Assume that all jobs have no more than c operations. Also, we assume that each operation is equally likely to be performed on any of the m machines. Then, the running time of the algorithm is $\sum_{j=1}^{n} \sum_{l=1}^{c} ((j-1)c/m + 1)O((j-1)c + l) = O((n/m)n^2c^2)$. We find that in a real-world job shop, n/m tends to be large.

Next we consider the settings for R1, R2, R3, and R4, which greatly impact the quality of solutions constructed by the insertion algorithm. R1 can be devised, based on job information such as deadline, total required machine time (the sum of processing times), etc. Let us take a

perspective that is backward in time for scheduling jobs. Then, R1 can be any of the following five choices:

1. LAD - Descending in order of job deadline
2. SMUT - Ascending in order of total machine time
3. LT - Descending in order of total machine time
4. LPT - Descending in order of job deadline less total machine time
5. LADαMT - Descending in order of job deadline less α times total machine time

For R2, we can start scheduling operations of a job either from the last to the first (denoted by LF), or in the reverse order (denoted by FF). For the tie-breaking rule R4, we can select either the left-most (LM) or the right-most (RM) machine position amongst those that yield the highest value with respect to the metric in use. Finally, we discuss the choice of R3. Let $\mathcal{G}_1$ denote $(\mathcal{O}, \mathcal{A}_0 \cup (\cup_{i=1}^{l} \mathcal{R}(u, o_i))$. Here, we choose $\sum_j \mu_j \rho(f_j, o_t; \mathcal{G}_1)$ for R3. Clearly, this function is non-decreasing as more and more operations are inserted. This choice of R3 performs well, but it is possible that there are better choices.

A few more words need to be said about the intuition behind these rules. First, Rule R1=LD puts us into a perspective of time horizon, i.e., we try to push jobs backwards from their deadlines. Hence, it seems that the choice of R2 that better matches R1=LD should be R2 =LF. Second, Rule R1=SMT is based solely on the total machine time. A rule of thumb is that getting "small" jobs through the system as quickly as possible generally helps to reduce work-in-process (WIP). However, for the sake of simplicity, R1=SMT ignores the effect of manufacturing interference on the actual amount of time a job will spend in the system, and the fact that jobs are due over a period of time. R1=LMT is its opposite. Third, Rules R1=LDMT and R1=LDαMT somewhat resemble the Material Requirements Planning (MRP) System approach to shop scheduling. Both take into account job deadlines and to some extent, the amount of time a job will spend in the system, which could be significantly longer than the total machine time due to manufacturing interference. In R1=LDαMT, the multiple α is supposed to be determined empirically.

A shifting bottleneck procedure

The Shifting Bottleneck Procedure (SBP) is introduced by Adams, Balas, and Zawack (1988) for solving CJSSP. We will adapt this procedure and apply it to **JSTIMP**.

Let us give a brief outline of the SBP for the makespan minimization problem. Let $\mathcal{M}_0$ denote the set of sequenced machines, i.e., those machines on which operations have already been sequenced. Initially, $\mathcal{M}_0 = \emptyset$. In each iteration, a processing order of operations on each unsequenced machine is determined by solving a relaxed single-machine problem to optimality, with the processing orders on the sequenced machines fixed and the capacity on the rest of the unsequenced machines assumed to be infinite. The machine whose relaxed single-machine problem yields the maximum optimal makespan is identified as the *bottleneck* (say, machine k). The processing order that has attained the minimum makespan on the bottleneck machine is accepted. $\mathcal{M}_0$ is set to $\mathcal{M}_0 \cup \{k\}$. Next, the processing order on each machine in $\mathcal{M}_0$ is, in turn, nullified and then recalculated by solving a relaxed single-machine problem similar to the above problem, with

the processing orders fixed on the rest of the machines in $\mathcal{M}_0$. During intermediate iterations, this step is repeated three times; and, when the last machine is sequenced it is carried out until no more improvement in the makespan is possible.

With the total inventory objective, we revise the algorithm and get a shifting bottleneck procedure:

A Shifting Bottleneck Procedure for JSTIMP

Step 0. Set $\mathcal{M}_0 = \emptyset, \Omega_0 = \emptyset$.

Step 1. Identify the bottleneck $k \in \mathcal{M} \backslash \mathcal{M}_0$ such that $\sum_j \mu_j \rho(f_j, o_t; \mathcal{A} \cup \Omega_0 \cup \mathcal{R}_k)$ attains the minimum, where $\mathcal{R}_k$ solves $\Pi'(\mathcal{O}, \mathcal{A} \cup \Omega_0; \mathcal{R}_k)$. Set $\mathcal{M}_0 = \mathcal{M}_0 \cup \{k\}, \Omega_0 = \Omega_0 \cup \mathcal{R}_k$.

Step 2. For each $k \in \mathcal{M}_0$, reoptimize the operation sequence by solving $\Pi'(\mathcal{O}, \mathcal{A} \cup (\Omega_0 \backslash \mathcal{R}_k; \mathcal{R}'_k)$. Set $\mathcal{R}_k = \mathcal{R}'_k$. Repeat this step 3 times if $\mathcal{M}_0 \neq \mathcal{M}$; else, repeat until there is no further improvement.

Step 3. If $\mathcal{M}_0 \neq \mathcal{M}$, go to 1; else, *EXIT*.

The running time of SBP depends on the time spent solving the relaxed single-machine problem. For instance, if we assume that there is no job reentrancy and $\mu_j^W = \mu_j^F = 1$ for all $j = 1, \ldots, n$, then $\Pi'(\mathcal{O}, \mathcal{A} \cup \Omega_0; \mathcal{R}_k)$ can be solved in $O(h^2)$ time (Chekuri et al. 2001), where h is the number of operations on machine k. Suppose that all jobs have no more than c operations. Assume that each operation is equally likely to be performed on any of the m machines while still maintaining the nonreentrancy assumption, and additionally make a simplifying assumption that in Step 2, reoptimization is always repeated three times even in the last iteration when $\mathcal{M}_0 = \mathcal{M}$. Then, the running time of SBP is $\sum_{k=1}^{m} (m - k + 1 + 3k) O((nc/m)^2) = O((m+1)n^2c^2/m)$. When m is large, the running time is equal to $O(n^2c^2)$. Compared with the insertion algorithm whose running time is $O((n/m)n^2c^2)$ (Section 4.1), SBP is clearly asymptotically faster, provided that n/m is large.

We have revised the subroutine for solving the relaxed single-machine problem in the above procedure so that the algorithm also works when there are reentrant jobs. However, the guaranteed 1.58 worst-case lower bound is no longer valid for the single-machine relaxation.

We have created three sets of test problems from the demand data and machine routing information provided by R&B Grinding. The numbers have been modified to disguise proprietary information. The number of operations per job varies and is generally far less than the total number of machines on the shop floor. All problems in the three test sets are characterized by $|\mathcal{M}|$—the number of machines, $|\mathcal{J}|$—the number of jobs, $|\mathcal{O}|$—the total number of operations, and additionally, whether "delay" operations or reentrant jobs are present.

An operation is a "delay" if the machine that this operation requires has infinite capacity. Jobs do not compete for machines on this type of operation, which can be started whenever necessary and will be completed in the amount of time equal to the processing time. Some jobs are reentrant meaning that they visit certain machines more than once.

Problem set 1 consists of four problems: SP1a, SP1b, SP2a, and SP2b. Table 1 lists the size of each problem. SP1a and SP1b represent a much shorter planning horizon than SP2a and SP2b. In SP1a and SP2a, there are both "delay" operations and reentrant jobs. By removing "delay" operations and reentrancy-causing operations in SP1a and SP2a, we come up with SP1b and SP2b, respectively. Two other problem sets are generated in a similar fashion. All numerical experiments

Table 1: Problem Set 1

Instance	$\|\mathcal{M}\|$	$\|\mathcal{J}\|$	$\|\mathcal{O}\|$
SP1a	71	157	422
SP1b	70	157	330
SP2b	140	1744	4922
SP2b	140	1744	3783

were conducted on an IBM Netvista Pentium III 733 MHz PC with 128 MB RAM, running the Windows 2000 operating system.

Combinations of different settings for R1, R2, and R4 are tested, and the variants of IA are labeled according to schema R1-R2-R4. For example, LAD-LF-RM designates the algorithm in which R1=LD, R2=LF, and R4=RM. The parameter α in R1=LDαMT rule is estimated for each test problem by formula $Z_0/\sum_j P_j$, where $P_j := \sum_{u \in \mathcal{O}_j} p(u)$ is the total processing time of job j. The value for Z_0 used in the calculation is the objective value of the solution found by algorithm LPT-LF-RM. For comparison, we also run a heuristic algorithm DD on these same instances. DD is based on a simple dispatching rule to schedule jobs backwards through time starting with jobs due in the most distant future. Operations of a selected job from the last to the first are placed at the beginning of the existing machine sequences.

Numerical results are presented in terms of both the objective value divided by the number of jobs, and CPU time in seconds. DD is extremely fast, as we would expect for any dispatching rule; however, it yields low-quality solutions. Because DD is too inferior in terms of solution quality to make a meaningful comparison with any of the IA variants that we have tested, the data points for DD are precluded. To some extent, we may infer that IA generally has much better performance than dispatching-rule-based methods.

A few observations can be made. First, in terms of solution quality, IA is a significant improvement over the dispatching rule DD at a moderate computational cost. In our numerical experiment, each IA variant solves all four test problems in a total of less than 18 minutes. Second, due-date-related algorithms with R2=LF yield the best performance with respect to solution quality. Algorithms in this class are LPT-LF-R4, LADαMT-LF-R4, and LAD-LF-R4. LPT-LF-LM attains the best solution quality; it realizes a 63% reduction in the objective value, compared to SMUT-FF-RM, which yields the poorest results. Third, all the algorithms in the plot incur comparable computational costs; in fact, all of them are within $\pm 3\%$ from the median. Finally, the best choice for R2 is LF, meaning that the last operation of a job should be scheduled first.

For SBP we adopt the same reoptimization scheme as in Adams et al. (1988), where a shifting bottleneck procedure is proposed for CJSSP. In our SBP, at most 3 reoptimization cycles are performed during intermediary iterations. In the last iteration when all machines are sequenced, SBP repeats reoptimization until no further improvement can be found. SBP is tested on problem sets 2 and 3, and results are compared with those of the best performing insertion algorithm.

We can make the following observations from the results. First, SBP uses only a fraction of the CPU time required by the Insertion Algorithm (IA). The solution quality achieved by SBP is superior: It outperforms IA in 12 out of the 15 problems in problem set 1, and 11 out of 15 problems in problem set 2. For problem set 1, the averages of the values in columns (3) and (6) are 9.15% and 74.16%, respectively. For problem set 2, the averages are 14.35% and 76.30%.

For the detailed computational results, please refer to **?**.

4.2 Multicommodity Distribution System Design (MDSD) Models

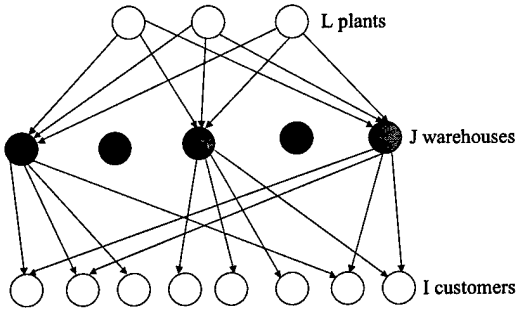

Figure 4: K products go from plants to warehouses to customers

Driven by the structures and data provided by our industrial partner, and similar models introduced in the literature (Simchi-Levi and Bramel 1997, Geoffrion and Graves 1974) we introduce the model below, which considers multiple manufacturing plants and multi-product families.

Features of our model include:

- A set of plants and customers are geographically dispersed in a region.
- Each customer experiences a demand for a variety of products that are manufactured at the plants. Products are shipped from plants to warehouses and then distributed to customers. For each shipping link and product there is a per unit shipping cost.
- Each customer's demand for each product is met from a single warehouse (single-source constraint).
- A given number of warehouses must be located in the distribution network from a list of potential sites. A fixed annual operating cost must be paid for each warehouse that is opened. Each warehouse has a capacity not to be exceeded.

We now give some notation that will be used throughout this paper. (see Figure 4)

The indices (sets) are as follows: $i(I)$ = customers, $j(J)$ = warehouses, $k(K)$ = products (commodities), $l(L)$ = plants. We also use I, J, K and L to denote cardinalities of those sets. The parameters are described in Table 2.

The problem variables are as follows:

$$Y_j = \begin{cases} 1 & \text{if a warehouse is opened at site } j \\ 0 & \text{otherwise,} \end{cases}$$

and

U_{ljk} = amount of product k from plant l to warehouse j for each $l \in L, j \in J,$ and $k \in K,$

parameter	description
c_{ljk}	unit shipping cost from plant l to warehouse j of product k
d_{jik}	unit shipping cost from warehouse j to customer i of product k
f_j	fixed cost of opening and operating a warehouse at site j
w_{ik}	demand of customer i for product k
v_{lk}	capacity of plant l for product k
q_j	capacity (in volume) of warehouse located at site j
s_k	volume of one unit of product k

Table 2: Problem Parameters

and

$$X_{jik} = \begin{cases} 1 & \text{if customer } i \text{ receives product } k \text{ from warehouse } j \\ 0 & \text{otherwise} \end{cases}$$

for each $j \in J, i \in I$ and $k \in K$.

The overall problem may now be stated. The objective function has three components. The first component is the transportation cost between the plants and warehouses, the second part of the objective function measures the transportation costs from warehouses to customers, and the last term of the objective is the fixed cost of locating and operating warehouses. Single supplier constraints guarantee that to every product/customer pair there is only one warehouse assigned. Warehouse capacity constraints ensure that the total amount of products shipped from warehouses does not exceed the capacities of the warehouses and that no shipments are made through closed warehouses. Conservation at warehouses constraints ensure that the amount of product arriving at a warehouse from plants equals the amount of products shipped from warehouses to customers. Plant capacity constraints guarantee that for every product, the amount of that product supplied by a plant does not exceed the production capacity of the plant for that product. The fixed number of warehouses constraint ensures that we locate exactly W warehouses. (Below we consider a strengthened model obtained by adding the constraints $X_{jik} \leq Y_j$ implied by the warehouse capacity constraints.) The algebraic description of the problem class is as follows:

$$\min \sum_{l=1}^{L}\sum_{j=1}^{J}\sum_{k=1}^{K} c_{ljk}U_{ljk} \qquad \text{Plant-warehouse shipping costs}$$

$$+\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} d_{jik}w_{ik}X_{jik} \qquad \text{Warehouse-customer shipping costs}$$

$$+\sum_{j=1}^{J} f_j Y_j \qquad \text{Warehouse fixed costs}$$

$$\text{s.t} \sum_{j=1}^{J} U_{ljk} \leq v_{lk} \quad \forall l \in L, k \in K \qquad \text{Plant capacity} \qquad (4)$$

$$\sum_{i=1}^{I}\sum_{k=1}^{K} s_k w_{ik} X_{jik} \leq q_j Y_j \quad \forall j \in J \qquad \text{Warehouse capacity} \qquad (5)$$

$$\sum_{i=1}^{I} w_{ik}X_{jik} = \sum_{l=1}^{L} U_{ljk} \quad \forall j \in J, k \in K \qquad \text{Conservation at warehouses} \qquad (6)$$

$$\sum_{j=1}^{J} Y_j = W \qquad \text{Fixed number of warehouses} \qquad (7)$$

$$\sum_{j=1}^{J} X_{jik} = 1 \quad \forall i \in I, k \in K \qquad \text{Single supplier} \qquad (8)$$

$$Y_j, X_{jik} \in \{0,1\} \quad \forall i \in I, j \in J, k \in K$$

$$U_{ljk} \geq 0 \quad \forall l \in L, j \in J, k \in K$$

Using a factorial design approach, we generated 32 problem sets of different sizes and subsequently generated ten additional larger problems by selectively increasing in some instances the number of customers to 250 or the number of products to 15. After randomly generating the data for each problem, we froze the data so that various solution strategies could be compared for each problem.

We used the AMPL 8.1 modeling language for mathematical programming to model the MDSD problems. The solver that we use in conjunction with AMPL 8.1 is CPLEX 8.1. We implemented a hybrid approach that we call C-NP-C (CPLEX-Nested Partitions-CPLEX) via looping constructs available in the AMPL modeling language (with CPLEX being used as the branch-and-cut solver), which represents a novel framework for the utilization of modeling-language/branch-and-cut software for the generation, evaluation, and coordination of information corresponding to appropriately partitioned global views of the original problem while taking good advantage of the structure of the problem class. The AMPL code of our C-NP-C implementation can be found in the Appendix B.

5. Conclusions

We introduced **JSTIMP**, whose objective is to minimize the total inventory. We discussed relevant complexity results of the problem and demonstrated how it could be approached with approximation algorithms through numerical examples. The reported work serves as a stepping stone for our future research, and further improvement can be made along several directions.

First, we will extend our current model, which may be denoted by $J|\bar{d}_j|WIP + FGI$, to include two additional constraints: *job ready times* and *machine ready times*. Some factors beyond the scope of scheduling decisions might mandate that the first operation of a job cannot start earlier than a certain time (job ready time). For example, the first operation of a job cannot start until the shipment of the required raw material arrives; neither can it start before the beginning of the next planning window. Furthermore, in a real-world manufacturing environment, scheduling problems are solved repeatedly to obtain updated schedules as the planning window moves forward through time. Most likely, some machines will be tied up with unfinished operations and thus will not be available for new operations at the beginning of the next planning window. Hence, operations contending for these machines have to wait until these machines become idle (machine ready time). Both job and machine ready times can be handled by introducing a release date r_j for each job j, and the extended model may be denoted as $J|r_j, \bar{d}_j|WIP + FGI$. This requires the solution of a general single-machine problem $1|r_j, \bar{d}_j|\sum w_j C_j$, which is subsequently addressed.

Second, instead of solving the relaxed single-machine problem approximately, we may try to use a branch-and-bound procedure to solve the problem to optimality. A higher solution quality is likely to be attained at a higher computational expense. In addition, a by-product of this approach is a lower bound on the optimal objective value for the original problem. This lower bound perhaps can be employed in composing a branch-and-bound procedure aimed directly at the original problem.

Third, the quality of the solution as well as the quality of the lower bound depends largely on how well the relaxed single-machine problem approximates $\Pi(\mathcal{O}, \mathcal{A} \cup \Omega; \mathcal{R}_k)$. What we have shown is simply one method of deriving a single-machine relaxation, and better relaxation schemes could likely be devised.

Finally, we would like to incorporate **JSTIMP** and its solution methods into our general manufacturing scheduling optimization framework based on the Nested Partitions method (Shi and Ólafsson 2000, Shi, Ólafsson, and Chen 2001).

As to the supply chain optimization problem, our computational experience demonstrates that while our new Lagrangian Relaxation (LR) approach often yields significantly better feasible solutions than stand-alone CPLEX for the more difficult problems in the test suite, the use of CPLEX within the NP framework of the C-NP-C hybrid generally outperforms even the LR methods in terms of quality of feasible solutions. Specifically, for the 20 harder problems in the test suite, C-NP-C obtains the best solution in 15 cases, the new LR approach is better in 3 cases, while stand-alone CPLEX is better in only 2 cases (one of these cases corresponds to a standard MIP model, and the other is the stronger model). Moreover, because of the effectiveness of the fast warm starting procedure employed by NP, the times required by C-NP-C to reach those highest quality solutions are generally only a few minutes (after which only rarely is a slightly better solution found). This is in strong contrast to all of the other methods which either typically produce poor quality solutions even after two hours (standard MIP approach) or which require initial solution of the relaxation of the strong MIP formulation, an initialization procedure that itself often

requires an hour or more (and, in one case, more than two hours) for the more difficult problems. Thus, the NP framework for the use of BC solvers has proven to be efficient, reliably fast, and effective in terms of providing high-quality solutions.

Our computational results demonstrate that for large-scale supply chain design problems, the Nested Partitions approach significantly outperforms both general-purpose combinatorial optimizers (such the branch-and-cut solver within CPLEX) and specialized approaches such as those based on Lagrangian relaxation. The Nested Partitions framework can effectively combine problem–specific heuristics with MIP tools (such as AMPL/CPLEX), so for this problem class we have developed an excellent warehouse ranking heuristic and used this heuristic to construct a "warm start" procedure followed by an effective biased sampling approach (using CPLEX to solve the MIPs corresponding to the samples) that uses this ranking and is guided by a global view of the problem. This research has also established the applicability of the AMPL/CPLEX modeling-language/solver combination as an excellent means for the implementation of the NP framework for global meta-hybrids, and thus represents a novel and successful use of these powerful software tools.

The Multicommodity Distribution System Design (MDSD) problem class investigated in this research has a constraint, based upon interaction with our industrial partner (and also quite common in the literature) requiring the selection of exactly W warehouses from the list of candidate locations. An interesting and equally-important alternative is the problem class in which at most W warehouses can be chosen to satisfy the constraints. The Lagrangian Relaxation and hybrid C-NP-C methods considered here may be suitably modified to handle that problem class.

We have demonstrated in this research that the looping constructs within well-known optimization modeling languages combined with good branch-and-cut solvers provide an excellent tool for implementing this approach to combinatorial optimization, and we plan to continue our development of such methods and software for a variety of challenging problem classes.

Reader is referred to our paper given in Appendix A for more details and numerical results of this research.

6. Personnel Supported

Personnel involved in this project are listed as follows:

Dr. Leyuan Shi, Principal Investigator
Dr. Sigurður Ólafsson, Visiting Professor
Adam Resnick, Ph.D. Student, graduated in December 2002
Yunpeng Pan, Ph.D. Student, graduated in May 2003
Shuli Men, Project Assistant
Edward Zhang, Graduate student

7. Publications

The following papers were published or submitted with the support of this grant.

1. Nembhard, H. B., L. Shi, and M. Aktan. 2003a. Real option design for product outsourcing. *The Engineering Economist* To appear.

2. Pan, Y. 2003a. An improved branch and bound algorithm for single machine scheduling with deadlines to minimize total weighted completion time. *Operations Research Letters*. To appear.

3. Shi, L., and S. Men. 2003. Optimal buffer allocation in production lines. *IIE Transactions* 35 (1): 1–10.

4. Fu, B.-R., L. Shi, and R. Suri. 2002. Analysis of departure times in discrete and continuous tandem production lines. *Journal of Discrete Event Dynamic Systems* 12 (2): 159–186.

5. Gong, W., and L. Shi. 2002. *Complex systems modeling and optimization*. Kluwer Academic Publishers.

6. Nembhard, H. B., and L. Shi. 2002. Real option design for quality control charts. *The Engineering Economist* 47 (1): 28–59.

7. Ólafsson, S., and L. Shi. 2002. Simulation optimization of discrete event systems. *Journal of Discrete Event Dynamic Systems* 12 (2): 211–240.

8. Shi, L., S. Ólafsson, and Q. Chen. 2001. An optimization framework for product design. *Management Science* 47 (12): 1681–1692.

9. Bozbay, M. 2003. Large-scale supply chain network optimization via nested partitions. Preliminary Exam, University of Wisconsin-Madison, Madison, WI.

10. Chalermdamrichai, V., D. Veeramani, and L. Shi. 2003. A nested partitions algorithm for optimal process planning on four-axis turning centers. *Submitted to Management Science*.

11. Nembhard, H. B., L. Shi, and M. Aktan. 2003b. The effect of implementation time lag on real options valuation. *Submitted to IIE Transactions*.

12. Pan, Y. 2003b, May. *Production scheduling for suppliers in the extended enterprise*. Ph. D. thesis, Department of Industrial Engineering, University of Wisconsin-Madison, Madison, WI.

13. Shi, L., R. R. Meyer, M. Bozbay, and A. J. Miller. 2003. Large-scale supply chain network optimization via a nested partitions framework. *To be submitted to INFORMS Journal on Computing*.

14. Shi, L., and Y. Pan. 2003. An effective technique for enhancing local search methods for the job-shop problem. *Submited to IEEE Transactions on Robotics and Automation*.

15. Pan, Y., and L. Shi. 2002. Hard instances of the one-machine sequencing problem. Under revision for European Journal of Operational Research.

16. Resnick, A. C. 2002, December. *Models for optimizing component safety stock levels in large scale assembly systems*. Ph. D. thesis, Department of Industrial Engineering, University of Wisconsin-Madison, Madison, WI.

8. Interactions/Transitions

a. Participation/presentations at meetings, conferences, seminars, etc.

1. Invited talk "Nested Partitions Methods and Its Application in Information Technology," ARO, December 2001.
2. Invited talk "Production Scheduling for Suppliers in the Extended Enterprise," INFORMS Annual Meeting, Atlanta, Georgia, October 2003.
3. "The Job-Shop Total Inventory Minimization Problem," INFORMS Annual Meeting, Atlanta, Georgia, October 2003.
4. "Hard Instances of the One-Machine Sequencing Problem," INFORMS Annual Meeting, San Jose, California, November 2002.
5. "The Nested Partitions Method for the Job-Shop Scheduling Problem," INFORMS Annual Meeting, Miami Beach, Florida, November 2001.
6. "A Real Options Design for Product Outsourcing," the 2001 Winter Simulation Conference, Washington D.C., 2001.
7. "Hybrid Meta-Heuristic for Combinatorial Optimization," NSF Grantee's Conference, January 2002.
8. Invited talk "Supply Chain Optimization" Tsinghua University, Beijing, June 2002.

b. Consultative and advisory functions to other laboratories and agencies, especially Air Force and other DoD laboratories.

- Panel reviewer for Operations Research, National Science Foundation.
- Reviewer for Army Research Office.
- Reviewer for the journal *Operations Research.*
- Editorial board member of the journal *Manufacturing and Service Operations Management.*
- Reviewer for the journal *Production and Operations Management.*
- Reviewer for the journal *Journal of Scheduling.*

9. New Discoveries, Inventions, or Patent Disclosures

None.

References

Adams, J., E. Balas, and D. Zawack. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34:391–401.

Ahmadi, R. H., and U. Bagchi. 1986a. Just-in-time scheduling in single machine systems. Working paper 85/86-4-21, Department of Management, University of Texas, Austin, TX.

Ahmadi, R. H., and U. Bagchi. 1986b. Single machine scheduling to minimize earliness subject to deadlines. Working paper 85/86-4-17, Department of Management, University of Texas, Austin, TX.

Ahmadi, R. H., and U. Bagchi. 1992. Minimizing job idleness in deadline constrained environments. *Operations Research* 40:972–985.

Balas, E. 1979. Disjunctive programming. *Annals of Discrete Mathematics* 5:3–51.

Chalermdamerichai, V. 1999. *A hybrid computer-intelligent, user-interactive, process plan optimizer for four-axis cnc turning centers*. Ph. D. thesis, Department of Industrial Engineering, University of Wisconsin-Madison, Madison, WI.

Chand, S., and H. Schneeberger. 1986. Single machine scheduling to minimize weighted earliness subject to no tardy jobs. *European Journal of Operational Research* 34:221–230.

Garey, M. R., and D. S. Johnson. 1979. *Computers and intractability: A guide to the theory of np-completeness*. San Francisco, CA: Freeman.

Geoffrion, A., and G. Graves. 1974. Multicommodity distribution system design by benders decomposition. *Management Science* 20:822–844.

Glover, F., C. McMilan, and B. Novick. 1989. Tabu search-Part I. *ORSA Journal on Computing* 1:190–206.

Glover, F., C. McMilan, and B. Novick. 1990. Tabu search-Part II. *ORSA Journal on Computing* 2:4–32.

Goldberg, D. E. 1989. *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Mendelson, E. 1997. *Introduction to mathematical logic*, 6–9. NY: Chapman & Hall.

Morton, T., and D. Pentico. *Heuristic scheduling systems: with application to production systems and project management*. NY: John Wiley & Sons.

Nawaz, M., J. E. E. Enscore, and I. Ham. 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA International Journal of Management Science* 11:91–95.

Ólafsson, S., and L. Shi. 2000. A method for scheduling in parallel manufacturing systems with flexible reources. *IIE Transactions* 32:135–146.

Pan, Y. 2003, May. *Production scheduling for suppliers in the extended enterprise, chapter 4*. Ph. D. thesis, Department of Industrial Engineering, University of Wisconsin-Madison, Madison, WI.

Roy, B., and B. Sussmann. 1964. Les problèmes d'ordonnancements avec contraintes disjonctives. Notes DS No. 9 bis, SEMA, Paris.

Shi, L., and S. Men. 2003. Optimal buffer allocation in production lines. *IIE Transactions* 35 (1): 1–10.

Shi, L., and S. Ólafsson. 2000. Nested partitions method for global optimization. *Operations Research* 48:390–407.

Shi, L., S. Ólafsson, and Q. Chen. 1999. New parallel randomized algorithms for the traveling

salesman problem. *Computers and Operations Research* 26:374–391.

Shi, L., S. Ólafsson, and Q. Chen. 2001. An optimization framework for product design. *Management Science* 47 (12): 1681–1692.

Simchi-Levi, D., and J. Bramel. 1997. *The logic of logistics: theory, algorithms, and applications for logistics management.* Springer Series in Operations Research.

Werner, F., and A. Winkler. 1995. Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics* 58:191–211.

Appendix A – Manuscript: Large-Scale Supply Chain Network Optimization via a Nested Partitions Framework

Large-Scale Supply Chain Network Optimization via a Nested Partitions Framework

Leyuan Shi • Robert R. Meyer† • Mehmet Bozbay
Andrew J. Miller

Department of Industrial Engineering, University of Wisconsin-Madison, Madison, WI 53706
†*Computer Sciences Department, University of Wisconsin-Madison, Madison, WI, 53706*
leyuan@engr.wisc.edu • rrm@cs.wisc.edu • mbozbay@wisc.edu • amiller@engr.wisc.edu

Abstract

Large-scale supply chain design problems are generally intractable with respect to standard mixed-integer programming (MIP) tools such as the direct application of general-purpose branch-and-cut (BC) commercial solvers such as CPLEX. In this research, we investigate a nested partitions (NP) framework that combines meta-heuristics with MIP tools (including branch-and-cut). We also consider a variety of alternative formulations and decomposition methods for this problem class. Our results show that our NP framework is capable of efficiently producing very high quality solutions to supply chain network design problems. For large-scale problems in this class, this approach is significantly faster and generates better feasible solutions than either CPLEX (applied directly to the given MIP) or the iterative Lagrangian-based methods that have generally been regarded as the most effective structure-based techniques for large-scale supply chain network optimization. The NP framework that we describe here first employs an MIP-based heuristic in order to generate an efficiency ranking for the available warehouses and then employs biased sampling (using this warehouse ranking and the NP global viewpoint) to guide the determination of additional restricted MIP subproblems that are then solved via CPLEX. This overall process is implemented via looping constructs available in the AMPL modeling language (with CPLEX being used as the branch-and-cut solver), and thus represents a novel framework for the utilization of modeling-language/branch-and-cut software for the generation, evaluation, and coordination of information corresponding to appropriately partitioned global views of the original problem while taking good advantage of the structure of the problem class. We also briefly discuss some other large-scale MIP problem classes for which this approach is expected to be very effective.

1 Introduction

In 2000, American companies spent $1 trillion, or about 10.1% of the United States gross national product (GNP), on supply related activities (Delaney, 2003). This total includes the cost of movement, storage and control of products across the supply chain, both within manufacturing plants and warehouses and between different components of the supply chain. Furthermore, the transportation costs accounted for $585 billion, or about 5.9% of the United States GNP. Unfortunately, this huge investment typically includes many unnecessary cost components due to redundant stock, inefficient transportation strategies, and other wasteful practices in the supply chain. In the face of increased competition in global markets, budget tightening in the supply chain has become essential.

In recent years, supply chain optimization has thus become a slogan among the world's leading manufacturers, distributors, and retailers. Supply chain optimization is a discipline that uses algorithmic tools, advanced information systems and innovative business processes to guide the movement of products from the point of origin to the point of consumption in the least amount of time, for the least cost.

Typically, we can categorize the decisions to be made in supply chain optimization into three categories based on horizons of their effects. They are strategic, tactical and operational level decisions. The strategic level decisions have long-term effects on a company. These decisions include the number, location, and capacity of warehouses and manufacturing plants, and the flow of material through these locations. The tactical level decisions are usually made once every quarter or once every year. These include purchasing and production decisions, inventory policies, and transportation strategies. The operational level decisions are made on a day-to-day basis. Scheduling, routing and truck loading are examples of decisions made at an operational level.

In this paper we address a strategic level issue: optimal supply chain network design. An optimized supply chain network can drastically improve supply chain performance in a variety of areas such as increased manufacturing throughput, reduced inventory costs, improved service levels, and better return on assets. This potential for improvement is generating a great deal of interest in supply network design optimization.

Our research work was motivated by a supply chain optimization project sponsored by a multi-billion dollar global industrial manufacturing firm in the United States. The company employs more than 20,000 people at hundreds of locations serving customers in more than 80 countries. Its supply chain (assembly and distribution network) is composed of manufacturing/assembly plants, and over 500 warehouse locations. There are multiple manufacturing sites that receive shipping orders, make assembly parts, and warehouse over 300,000 different components and parts. The network of fulfillment nodes (warehouses) supplies over 100,000 customer locations. These warehouse locations can be considered to be a stocking point of product.

From a computational viewpoint, supply chain network design problems are quite challenging

because they are combinatorially explosive - some of these optimization models have millions of variables and millions of constraints. The problem belongs to the class of NP-Complete problems (Garey and Johnson, 1979), which means any attempt to obtain an optimal solution for every instance of the problem will require unacceptable levels of computing time.

There are two principal technologies for large-scale supply chain optimization problems: 1) exact algorithms (such as linear-programming-based BC and constraint programming techniques) that are guaranteed to find optimal solutions within a finite (but possibly unacceptably large) amount of computation, and 2) heuristic algorithms that quickly find good, but not necessarily optimal, solutions. Exact algorithms based on relaxation are powerful, useful and vitally necessary in optimization. Unfortunately, these approaches also have some limitations:

- certain classes of problems remain difficult for these technologies (for example, product design and scheduling problems, and large-scale problems)
- in MIP models, the user must express all problem constraints in terms of linear relationships that usually require the introduction of many additional variables and constraints (this makes the model much more complex, and the corresponding continuous relaxation is often a very poor approximation to the original problem)
- the use of Branch & Cut and related general-purpose technologies limit the user's ability to incorporate domain knowledge into the solution procedure.

There are numerous heuristics for solving supply chain problems. However, rather than considering these heuristics in isolation (or employing a randomized multi-start mechanism), our Nested Partitions (NP) approach obtains maximum benefit from their availability by employing them within a global framework of partition-based strategies. We use the term global meta-heuristic for the approach resulting from such a framework since it repeatedly applies one or more heuristics within partitions of the original domain in order to maintain a global view of the problem. This global view allows a proof of optimality to be constructed in spite of the use of heuristics. (Of course, as with BC, such a demonstration of optimality may be computationally expensive, so termination with a solution of guaranteed high quality is, from a practical viewpoint, often a preferred alternative.) The term global meta-hybrid is used to denote the combination of a global meta-heuristic (such as NP) with lower-bounding techniques (such as linear programming). Such a hybrid allows synergistic coordination of the lower bound information with the upper bounds and feasible solution data from the heuristic.

Our research provides new global hybrid algorithms and software tools for large-scale supply chain optimization and extends the range of problems for which the hybrid randomized optimization approach significantly outperforms alternatives such as Branch & Cut, Lagrangian relaxation and heuristic search techniques not based on partitioning. We tested our hybrid algorithm with different

size test problems - the largest test problem having 15 products, 10 plants, 100 possible warehouse locations and 250 customers.

The remainder of the paper is organized as follows: section 2 describes the Multicommodity Distribution System Design (MDSD) Models that are the applications focus of this paper, section 3 deals with existing solution techniques with emphasis on branch-and-cut and Lagrangian relaxation approaches, our new methods are presented in section 4, and computational results comparing existing methods with the new techniques are given in section 5. Conclusions and further research directions for our new approach conclude the paper.

2 Multicommodity Distribution System Design (MDSD) Models

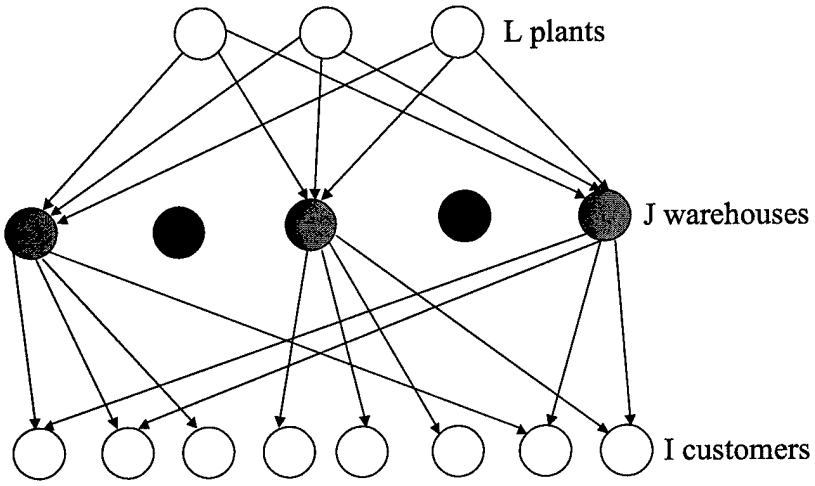

Figure 1: K products go from plants to warehouses to customers

Driven by the structures and data provided by our industrial partner, and similar models introduced in the literature (Simchi-Levi and Bramel, 1997), (Geoffrion and Graves, 1974) we introduce the following model, which considers multiple manufacturing plants and multi-product families.

2.1 MDSD Model MIP Formulation

Our model has the following features:

- A set of plants and customers are geographically dispersed in a region.
- Each customer experiences a demand for a variety of products that are manufactured at the plants. Products are shipped from plants to warehouses and then distributed to customers.

parameter	description
c_{ljk}	unit shipping cost from plant l to warehouse j of product k
d_{jik}	unit shipping cost from warehouse j to customer i of product k
f_j	fixed cost of opening and operating a warehouse at site j
w_{ik}	demand of customer i for product k
v_{lk}	capacity of plant l for product k
q_j	capacity (in volume) of warehouse located at site j
s_k	volume of one unit of product k

Table 1: Problem Parameters

For each shipping link and product there is a per unit shipping cost.

- Each customer's demand for each product is met from a single warehouse (single-source constraint).
- A given number of warehouses must be located in the distribution network from a list of potential sites. A fixed annual operating cost must be paid for each warehouse that is opened. Each warehouse has a capacity not to be exceeded.

We now give some notation that will be used throughout this paper. (see Figure 1)

The indices (sets) are as follows: $i(I)$ = customers, $j(J)$ = warehouses, $k(K)$ = products (commodities), $l(L)$ = plants. We also use I, J, K and L to denote cardinalities of those sets. The parameters are described in Table 1.

The problem variables are as follows:

$$Y_j = \begin{cases} 1 & \text{if a warehouse is opened at site } j \\ 0 & \text{otherwise,} \end{cases}$$

and

U_{ljk} = amount of product k from plant l to warehouse j for each $l \in L, j \in J,$ and $k \in K,$

and

$$X_{jik} = \begin{cases} 1 & \text{if customer } i \text{ receives product } k \text{ from warehouse } j \\ 0 & \text{otherwise} \end{cases}$$

for each $j \in J, i \in I$ and $k \in K$.

The overall problem may now be stated. The objective function has three components. The first component is the transportation cost between the plants and warehouses, the second part of the objective function measures the transportation costs from warehouses to customers, and the last term of the objective is the fixed cost of locating and operating warehouses. Single supplier constraints guarantee that to every product/customer pair there is only one warehouse assigned. Warehouse capacity constraints ensure that the total amount of products shipped from warehouses does not exceed the capacities of the warehouses and that no shipments are made through closed warehouses. Conservation at warehouses constraints ensure that the amount of product arriving at a warehouse from plants equals the amount of products shipped from warehouses to customers. Plant capacity constraints guarantee that for every product, the amount of that product supplied by a plant does not exceed the production capacity of the plant for that product. The fixed number of warehouses constraint ensures that we locate exactly W warehouses. (Below we consider a strengthened model obtained by adding the constraints $X_{jik} \leq Y_j$ implied by the warehouse capacity constraints.) The algebraic description of the problem class is as follows:

$$\min \sum_{l=1}^{L}\sum_{j=1}^{J}\sum_{k=1}^{K} c_{ljk}U_{ljk} \quad \text{Plant-warehouse shipping costs}$$

$$+\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} d_{jik}w_{ik}X_{jik} \quad \text{Warehouse-customer shipping costs}$$

$$+\sum_{j=1}^{J} f_j Y_j \quad \text{Warehouse fixed costs}$$

$$\text{s.t} \quad \sum_{j=1}^{J} U_{ljk} \leq v_{lk} \quad \forall l \in L, k \in K \quad \text{Plant capacity} \tag{2.1}$$

$$\sum_{i=1}^{I}\sum_{k=1}^{K} s_k w_{ik} X_{jik} \leq q_j Y_j \quad \forall j \in J \quad \text{Warehouse capacity} \tag{2.2}$$

$$\sum_{i=1}^{I} w_{ik} X_{jik} = \sum_{l=1}^{L} U_{ljk} \quad \forall j \in J, k \in K \quad \text{Conservation at warehouses} \tag{2.3}$$

$$\sum_{j=1}^{J} Y_j = W \quad \text{Fixed number of warehouses} \tag{2.4}$$

$$\sum_{j=1}^{J} X_{jik} = 1 \quad \forall i \in I, k \in K \quad \text{Single supplier} \tag{2.5}$$

$$Y_j, X_{jik} \in \{0,1\} \quad \forall i \in I, j \in J, k \in K$$

$$U_{ljk} \geq 0 \quad \forall l \in L, j \in J, k \in K$$

3 Existing Solution Techniques for MDSD Models

Location problems cover a wide range of problems in which the goal is to locate a set of facilities in a distribution network while satisfying the given constraints. Location models (even the most basic ones) are generally computationally intractable for large problem instances. One of the major reasons for the recent extensive interest in formulation and implementation of location models is due to the advances in algorithms and computer technologies that allow larger instances to be solved in reasonable times.

In the MDSD models, the goal can be the selection of a predetermined number of warehouses that minimize cost (the focus of this paper) or simply finding an optimal set of warehouses to minimize the total cost. Also, the problem may require that each customer is supplied by exactly one warehouse for each product (single-source) (Neebe and Rao, 1983) or may allow splitting of the

demand for a product by a customer among a number of warehouses (multiple-source)(Simchi-Levi and Bramel, 1997). The assumption of a customer getting delivery for a particular product from only one warehouse does not preclude solutions in which a customer gets shipments from other warehouses, but these shipments must be for different products. On the other hand, it is assumed that the warehouse can receive shipments from any plant for any product.

The literature on warehouse location problems is very extensive, dealing with different models and different scenarios. (Cordeau et al., 2001) proposed assigning locomotives and cars to trains as a multicommodity network flow-based model. They formulated the problem as a mixed-integer problem, and solved it by a branch-and-bound method that relaxes some of the integrality constraints. At each node of the tree, they solve a MIP problem by Benders decomposition. (Lee, 1993) proposed a heuristic based on Benders decomposition and Lagrangian Relaxation for a multicommodity capacitated single-layer (no plants) facility location problem with a choice of facility type in which each facility type offers a different capacity for a particular product with different fixed setup costs. The largest test problem had 200 possible warehouse locations, 200 customers, 20 products, and 10 facility types.

However, not much work has been done on the two-layer warehouse location models (Hindi and Basta, 1994). One of the classical papers in solving locations problems is (Geoffrion and Graves, 1974), which solves the MDSD problem using Benders decomposition. Their largest test problem had 5 products, 3 plants, 67 possible warehouse locations and 127 customer zones. The focus of this research will be these type of problems where multiple layers and multiple product types are considered.

Recent literature on facility location models offers several additional solution techniques. These include but not are limited to branch & bound and cutting plane algorithms ((Hindi and Basta, 1994), (Aardal, 1998)), Lagrangian Relaxation techniques ((Pirkul and Jayaraman, 1996), (Klincewicz and Luss, 1986)), simulated annealing ((Mathar and Niessen, 2000)), and Tabu search algorithms ((Crainic et al., 1996), (Delmaire et al., 1999)). Branch & Bound (BB), Branch & Cut, and Lagrangian Relaxation approaches are currently the most widely applied solution techniques in this area and will be discussed in the following sections.

3.1 Branch & Bound and Branch & Cut Algorithms

The Branch & Bound algorithm can be applied to large and numerically difficult Mixed Integer Programming (MIP) problems. A Mixed Integer Program is essentially a Linear Program (LP) with integrality restrictions on one or more variables. The Branch & Bound algorithm guarantees optimality (assuming the problem has an optimal solution) if allowed to run to completion, which might require unacceptably large solution times.

The algorithm keeps track of related LP subproblems in a tree format. The algorithm begins by relaxing the original problem (the root node in the tree) and solving it. The relaxed problem

is an LP in which all integrality constraints are dropped from the original problem. The solution obtained from the relaxation will be the optimal solution if integrality restrictions are not violated. In most cases, however, the LP relaxation is likely to yield fractional values for certain integer variables. Provided that an integer optimal solution has not been found, the algorithm proceeds by selecting a fractional variable for branching. New subproblems are then generated by fixing this variable to 0 and to 1 (only binary variables considered here). The procedure continues recursively by solving the LP relaxations of the new subproblems.

In the Branch & Cut algorithm, if the solution to the relaxation has one or more fractional variables, cuts will be generated. Cuts are constraints that cut away areas of the feasible region of the relaxation that contain fractional solutions but exclude no integer solutions. There are several types of cuts that can be utilized. If the solution to the relaxation still has one or more fractional-valued integer variables after the cuts are applied, then it branches on a fractional variable to generate two new subproblems.

We used the AMPL 8.1 modeling language for mathematical programming to model the MDSD problems. The solver that we use in conjunction with AMPL 8.1 is CPLEX 8.1.

The reader is referred to (Mitchell, 2000) for a discussion of Branch-and-Cut algorithms for combinatorial optimization problems. (Hindi and Basta, 1994) obtained feasible solutions for the multiple-source MDSD problem using branch-and-bound, and obtained lower bounds using Dantzig-Wolfe decomposition. Their largest test problem had 3 products, 10 plants, 15 possible warehouse locations and 30 customers.

3.2 Lagrangian Relaxation (Conventional Approach)

Many hard integer-programming problems can be viewed as easy problems complicated by a relatively small set of side constraints (Wolsey, 1998). Dualizing the side constraints produces a Lagrangian problem that is easy to solve and whose optimal value is a lower bound (for minimization problems) on the optimal value of the original problem. The Lagrangian problem can thus be used in place of a linear programming relaxation to provide bounds in a Branch & Bound (BB) algorithm (Fisher, 1981). (Beasley, 1993) demonstrates the use of Lagrangian relaxation (LR) methods to solve various location problems. In our research, we consider two different LR approaches. In the first, which we term the conventional approach, we relax single supplier constraints and conservation at warehouse constraints, a common practice in the literature ((Pirkul and Jayaraman, 1996), (Simchi-Levi and Bramel, 1997)). (In section 4, we introduce a new LR approach for this problem.) In particular, we relax single supplier constraints (with multipliers λ_{ik}) and conservation at warehouse constraints (with multipliers θ_{jk}). The resulting problem is

$$\min \sum_{l=1}^{L}\sum_{j=1}^{J}\sum_{k=1}^{K} c_{ljk}U_{ljk} + \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} d_{jik}w_{ik}X_{jik} + \sum_{j=1}^{J} f_jY_j +$$

$$\sum_{j=1}^{J}\sum_{k=1}^{K}\theta_{jk}\left[\sum_{i=1}^{I}w_{ik}X_{jik}-\sum_{l=1}^{L}U_{ljk}\right]+\sum_{i=1}^{I}\sum_{k=1}^{K}\lambda_{ik}\left[1-\sum_{j=1}^{J}X_{jik}\right]$$

subject to the following constraints: plant capacity (U variables), warehouse capacity (X, Y variables), fixed number of warehouses (Y variables).

Because the relaxation of conservation of flow decouples warehouse shipments from plant shipments, this problem can be decomposed into two separate problems P_{PW} and P_{WC}:

- Let Z_{PW} be the optimal value of the ***plant-warehouse subproblem*** P_{PW}:

$$Z_{PW} = \min \sum_{l=1}^{L}\sum_{j=1}^{J}\sum_{k=1}^{K}\left[c_{ljk}-\theta_{jk}\right]U_{ljk}$$

$$\text{s.t.} \sum_{j=1}^{J}U_{ljk} \le v_{lk} \qquad \forall l \in L, k \in K$$

$$U_{ljk} \ge 0 \qquad \forall l \in L, j \in J, k \in K$$

- Let Z_{WC} be the optimal value of the ***warehouse-customer subproblem*** P_{WC}:

$$Z_{WC} = \min \sum_{j=1}^{J}\sum_{i=1}^{I}\sum_{k=1}^{K}\left[d_{jik}w_{ik}-\lambda_{ik}+\theta_{jk}w_{ik}\right]X_{jik}+\sum_{j=1}^{J}f_jY_j$$

$$\text{s.t.} \sum_{i=1}^{I}\sum_{k=1}^{K}s_kw_{ik}X_{jik} \le q_jY_j \qquad \forall j \in J$$

$$\sum_{j=1}^{J}Y_j = W$$

$$Y_j, X_{jik} \in \{0,1\} \qquad \forall i \in I, j \in J, k \in K$$

Since there is no coupling between warehouses except for the cardinality constraint $\sum_{j=1}^{J}Y_j = W$, the preceding warehouse-customer problem can be solved by considering a set of J single warehouse problems. Specifically, for every warehouse j , the following knapsack problem (corresponding to $Y_j = 1$) with IK items is solved to obtain a "value" for warehouse j:

$$Z_{WC}^{j} = \min \sum_{i=1}^{I}\sum_{k=1}^{K}\left[d_{jik}w_{ik}-\lambda_{ik}+\theta_{jk}w_{ik}\right]X_{jik}+f_j$$

$$\text{s.t.} \sum_{i=1}^{I}\sum_{k=1}^{K}s_kw_{ik}X_{jik} \le q_j$$

$$X_{jik} \in \{0,1\} \quad \forall i \in I, k \in K,$$

and the optimal solution to P_{WC} is then obtained by concatenating the best W solutions of these separate problems (Simchi-Levi and Bramel, 1997) and closing the remaining warehouses. Then $Z_{\lambda,\theta} = Z_{PW} + Z_{WC} + \Sigma_i \Sigma_k \lambda_{ik}$ is a lower bound to the original problem. To approximately maximize $Z_{\lambda,\theta}$, we use a subgradient algorithm.

Since solving the Lagrangian subproblems usually results in solutions that are not feasible the original problem, many implementations of this technique use a heuristic to obtain feasible solutions from relaxed solutions. (Pirkul and Jayaraman, 1996) used Lagrangian Relaxation to obtain lower bounds for the MDSD problem while employing a heuristic to obtain feasible solutions at every iteration. Their largest test problem had 10 products, 10 plants, 30 possible warehouse locations and 75 customers.

In our implementation of the Lagrangian approach as described below, we employed a different heuristic (Simchi-Levi and Bramel, 1997). This heuristic first chooses the W warehouses defined by the best W solutions of the knapsack problems for each warehouse defined above, and then solves a modified version of the original problem with Y_j fixed to 1 for the W best warehouses and Y_j fixed to 0 for the others. While this solution is not guaranteed to be feasible, in our experience we have found that it nearly always feasible. The capacities of the warehouses are generated in such way that in the feasible solutions warehouse capacities are utilized at around 85%. In any event, it is not necessary that a feasible solution be produced at every LR iteration. Solving this MIP typically takes only a fraction of time required for solving the original problem, and never more than a few seconds for the problems in our test set.

4 New Approaches for MDSD

In this section, we introduce three new approaches to the MDSD problem. In subsection 4.1, a stronger formulation for the MIP is given; subsection 4.2 introduces a new Lagrangian Relaxation formulation related to this stronger formulation. In subsection 4.3, we describe a new hybrid approach that uses the looping constructs of the AMPL modeling language to construct MIP subproblems (solved via CPLEX) that correspond to an implementation of the Nested Partitions (NP) framework. Computational results demonstrating the superiority of this NP framework are given in section 5.

4.1 MIP CPLEX 8.1 (Strong Formulation)

The computational results below demonstrate that BC applied to the MIP formulation of the MDSD problem given in subsection 2.1 provides poor quality lower bounds for the large test problems solved. To obtain improved lower bounds, taking advantage of the fact that the variable Y_j associated with warehouse j must be equal to 1 if there is any shipment of any product from this

warehouse to any customer, we include the following set of additional constraints in the problem formulation.

$$X_{jik} \leq Y_j \qquad \forall i \in I, j \in J, k \in K \tag{4.1}$$

The addition of these constraints results in a much tighter LP relaxation. For example, with these constraints it is no longer necessary to impose integrality on the Y variables, since this is guaranteed in any optimal solution in which the X variables are integer.(Computationally, it is useful to maintain integrality of the Y variables for branching purposes.) Moreover, the lower bounds obtained from the corresponding LP relaxation are much better than the BC results for the original formulation. However, including these additional constraints causes the times taken to solve the LP relaxation at the root node to be considerably larger than for the original formulation. These issues are considered further in the section on computational results (see Table 5). We also note that the LR approach of section 3.2 can be started with duals from the LP relaxation of the strong formulation and will then provide even better lower bounds.

Proposition 1 *The lower bound obtained by maximizing the function $Z_{\lambda,\theta}$ introduced in section 3.2 is at least as strong as that obtained by solving the strong formulation LP relaxation.*

Proof: Consider an equivalent version of P_{WC} that includes the cut constraints (4.1). Let $\bar{Z}$ be the optimal value of the strong formulation LP relaxation and let $\bar{\lambda}$ and $\bar{\theta}$ be the optimal dual variables associated with constraints (2.5) and (2.3), respectively, in that relaxation. Let $\hat{Z}$ be the lower bound corresponding to $Z_{PW} + \Sigma_i \Sigma_k \bar{\lambda_{ik}}$ plus the optimal value of the LP relaxation of the equivalent version of P_{WC} . Complementary slackness ensures that the lower bound $\hat{Z}$ will be equal to $\bar{Z}$. Thus, when we impose integrality in P_{WC} and delete the redundant constraints (4.1), we will have $Z_{\bar{\lambda},\bar{\theta}} \geq \bar{Z}$, with strict inequality being likely. Moreover, maximizing $Z_{\lambda,\theta}$ will likely yield an even better lower bound than $Z_{\bar{\lambda},\bar{\theta}}$. ■

4.2 New Lagrangian Relaxation Method

The inclusion of the constraints (4.1) allows us to derive a new Lagrangian Relaxation approach based on decoupling the warehousing decisions from the rest of the problem. In particular, we relax warehouse capacity constraints (2.2) (with multipliers α_j) and strong formulation cut constraints (4.1) (with multipliers β_{jik}). The resulting problem is

$$\min \sum_{l=1}^{L}\sum_{j=1}^{J}\sum_{k=1}^{K} c_{ljk}U_{ljk} + \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} d_{jik}w_{ik}X_{jik} + \sum_{j=1}^{J} f_jY_j +$$
$$\sum_{j=1}^{J} \alpha_j \left[\sum_{i=1}^{I}\sum_{k=1}^{K} s_k w_{ik} X_{jik} - q_j Y_j\right] + \sum_{j=1}^{J}\sum_{i=1}^{I}\sum_{k=1}^{K} \beta_{jik}\left[X_{jik} - Y_j\right]$$

subject to the following constraints: plant capacity (U variables), conservation at warehouses (X, U variables), fixed number of warehouses, single supplier (X variables).

Since the warehouse Y_j variables are now decoupled from the shipping variables, this problem can be decomposed into two separate problems P_S and P_W. They are the following:

- Let Z_S be the optimal value of the ***shipping subproblem*** P_S:

$$Z_S = \min \sum_{l=1}^{L}\sum_{j=1}^{J}\sum_{k=1}^{K} c_{ljk}U_{ljk} + \sum_{j=1}^{J}\sum_{i=1}^{I}\sum_{k=1}^{K} \left[d_{jik} + \alpha_j s_k\right] w_{ik}X_{jik} + \sum_{j=1}^{J}\sum_{i=1}^{I}\sum_{k=1}^{K} \beta_{jik}X_{jik}$$

$$\text{s.t.} \sum_{j=1}^{J} X_{jik} = 1 \quad \forall i \in I, k \in K$$

$$\sum_{i=1}^{I} w_{ik}X_{jik} = \sum_{l=1}^{L} U_{ljk} \quad \forall j \in J, k \in K$$

$$\sum_{j=1}^{J} U_{ljk} \leq v_{lk} \quad \forall l \in L, k \in K$$

$$X_{jik} \in \{0,1\} \quad \forall j \in J, i \in I, k \in K$$

- Let Z_W be the optimal value of the ***warehouse subproblem*** P_W:

$$Z_W = \min \sum_{j=1}^{J} \left[f_j - \alpha_j q_j\right] Y_j - \sum_{j=1}^{J}\sum_{i=1}^{I}\sum_{k=1}^{K} \beta_{jik}Y_j$$

$$\text{s.t.} \sum_{j=1}^{J} Y_j = W$$

$$Y_j \in \{0,1\} \quad \forall j \in J$$

Then $Z_{\beta,\alpha} := Z_S + Z_W$ is a lower bound to the original problem.

Proposition 2 *The lower bound obtained by maximizing the function $Z_{\beta,\lambda}$ is at least as strong as that obtained by solving the strong formulation LP relaxation.*

Proof: Consider $\bar{Z}$, the optimal value of the strong formulation LP relaxation. If we choose $\bar{\alpha}$ and $\bar{\beta}$ to be the optimal dual variables associated with constraints (2.2) and (4.1), respectively and compute a lower bound *without* imposing integrality in Z_W, complementary slackness ensures that this lower bound will be equal to $\bar{Z}$. When we impose integrality in the formulation, we will thus

have $Z_{\bar{\beta},\bar{\alpha}} \geqslant \bar{Z}$, with strict inequality being likely. Moreover, maximizing $Z_{\beta,\alpha}$ will likely yield an even better lower bound than $Z_{\bar{\beta},\bar{\alpha}}$. ■

To approximately maximize $Z_{\beta,\alpha}$, as with the conventional LR approach, we use a standard subgradient algorithm. To generate feasible solutions, we can simply take the optimal solution to Z_W and solve the original problem with Y_j fixed to 1 for these warehouses and Y_j fixed to 0 for the rest.

We note that the shipping subproblem is considerably more complicated to solve than either of the subproblems of the conventional LR method. For this reason, we expect one evaluation of $Z_{\lambda,\theta}$ to take less time than one evaluation of $Z_{\beta,\alpha}$; that is, one iteration of the conventional LR method will likely take less time than one iteration of the new method. However, the added subproblem complexity means that maximizing $Z_{\beta,\alpha}$ will likely provide a better lower bound than maximizing $Z_{\lambda,\theta}$. That is, relaxing less yields an LR method that can be expected to provide a tighter lower bound that will compensate for the smaller number of iterations that can be performed in a given amount of time. Our computational results indicate that this often happens (see Table 7).

An additional advantage of the new formulation is that, at each iteration, the set of warehouses used in the construction of the feasible solutions (that provide upper bounds) depends only on the current Lagrangian multipliers (and original problem costs), and thus the composite warehouse costs in P_W may be thought of as a warehouse ranking. If the Lagrange multipliers are good in the sense that the composite costs correspond to a good warehouse ranking, a good feasible solution will be produced. In the conventional LR method, the warehouse set used to generate a feasible solution is derived from P_{WC}, but the customer allocation values that play a key role in determining the optimal warehouse set in P_{WC} are not subsequently re-used in the upper bounding process (since these values may be conflicting in the overall problem corresponding to the warehouses selected by the procedure). Since the fact that this set of warehouses is the best set for P_{WC} depends not only on the current composite warehouse costs, but also on the composite customer allocation costs in P_{WC}, one might expect that not being able to re-use a key part of the LR solution means that such an approach is inferior to one in which all of the solution of one of the LR subproblems is re-used. This expectation is borne out by our computational results (Table 7).

4.3 Hybrid CPLEX-Nested Partitions-CPLEX Approach (C-NP-C)

We first sketch a generic nested partitions procedure (Shi and Ólafsson, 2000) and then describe an implementation of Nested Partitions (NP) that takes advantage of the structure of the supply chain problem.

Consider the following combinatorial optimization problem, where Θ is a finite set: $min_{x\epsilon\Theta} f(x)$. (In the context of the MDSD problem class, Θ corresponds to the set of values of the binary warehouse selection variables that satisfy the warehouse cardinality constraints (2.4), and f(x)

is the optimal value of the supply chain problem with the warehouse set determined by x.) The

NP framework can be briefly described as follows. In each iteration of the algorithm we assume that we have a partition and a region (subset belonging to the partition) of Θ that is considered the *most promising* (initially, this is Θ). We then partition this most promising region into sub-regions (for example, M sub-regions) and aggregate the other regions (the aggregate is termed the complementary region) into one region. At each iteration, we thus consider M + 1 disjoint subsets of the feasible region Θ. Each of these M + 1 regions is sampled using some random sampling scheme, and for each region a *promise index* is calculated. (A variety of approaches may be used to calculate the promise index (see (Shi and Ólafsson, 2000)), but in the results below we simply used the objective value of the best sample in the region. Other possibilities in this context would involve functions that took into account both the best sample, which provides an upper bound for the region, and the LP relaxation value, which provides a lower bound for the region.) These promise indices are then compared to determine which region is the most promising in the next iteration. If one of the child subregions is found to have the best promise index, this subregion becomes the most promising region. (In an ideal instance, an unbroken sequence of child regions containing an optimal solution would be obtained, yielding a depth-first search that quickly leads to an optimal solution.) If, on the other hand, the surrounding region is found to have the best promise index, a subset of the surrounding region becomes the most promising. The new most promising region is then partitioned and sampled in a similar fashion. Since Θ is finite, in a finite number of iterations we reach regions that contain only a single sample point, and for such a region the sampling procedure generates the single sample point, and no further consideration of that region is required. As the algorithm evolves, a sequence of most promising regions $\sigma(k)$ will be generated, where $\sigma(k)$ is the most promising region in the k^{th} iteration. In (Shi and Ólafsson, 2000), it is shown that $\sigma(k)$ is a Markov chain with all the global optima as its absorbing states. Thus, global convergence is guaranteed. The only requirement on the random sampling procedure is that each point in the region has a positive probability of being selected. Hence there is much flexibility in selecting a sampling procedure.

A more formal description of a generic NP approach is as follows:

1. Partitioning

The first step is to partition the current most promising region into several subregions and aggregate the remaining regions into one region, called the complementary region. The partitioning strategy imposes a structure on the feasible region and is therefore very important for the rate of convergence of the algorithm. Ideally, we would like to see most of the good solutions clustered together in the a few subregions after the partitioning. This would allow the algorithm to quickly concentrate the search on these subsets of the feasible region.

2. Random Sampling

Next, we generate samples from the subregions and from the complementary region. This can be done in almost any fashion, provided that each solution in a given sampled region is selected with

a positive probability. Uniform sampling is one common way of sampling, but it is often worthwhile to incorporate special structures into the sampling procedure so that the sample quality can be improved. For example, we can use some weighted sampling scheme to generate a sample. The weight can be determined by carefully considering the system structure so that some solutions have a higher probability of being chosen than others.

3. Calculation of the Promise Index

For each region σ_j, $j = 1, 2, ..., M_{\sigma(k)} + 1$ define a promising index function, $I(\sigma_j), I : \Sigma \rightarrow R$, and calculate the *promise index.* For example, an ideal promise index function would yield the best performance value in the region.

$$I(\sigma) = \min_{\theta \in \sigma} f(\theta), \sigma \in \Sigma.$$

Since this index cannot in general be calculated for a region, we estimate the promise index $I(\sigma_j)$ by using N_j sample points:

$$\hat{I}(\sigma_j) = \min_{i=1,2,...,N_j} f(\theta^{j_i}), j = 1, 2, \ldots, M_{\sigma(k)} + 1.$$

Notice that $\hat{I}(\sigma_j)$ is a random variable. Other alternatives for $I(\sigma_j)$ can be found in (Shi and Ólafsson, 2000) and (Shi et al., 2001).

4. Backtracking

Determine the *new most promising index*

$$j_k \in \arg\min_{j=1,2,...,M_{\sigma(k)}+1} \hat{I}(\sigma_j), j = 1, 2, \ldots, M_{\sigma(k)} + 1.$$

If more than one region is equally promising, the smallest such index is chosen. The new most promising region $\sigma(k+1)$ is a child of the current most promising region unless the new most promising index corresponds to the complementary region, in which case backtracking is performed (for example, to the parent of the current most promising region or to the smallest subregion yet generated that contains the best sample point).

NP can be regarded as a generic algorithm capable of supporting a variety of partitioning techniques, sampling methods, promise indices, and backtracking rules. For example, two extreme alternatives of backtracking are to the parent of the promising region on the one hand, and, on the other hand, to the smallest subregion yet generated that contains the best sample point. These alternatives roughly correspond to depth-first and breadth-first (or best bound or estimate) BB strategies.

It should be noted that the promise index is defined on a sequence of sets. Therefore, we can incorporate any effective heuristic methods into the algorithm and use them to calculate the promise index function. This is possible because the only requirement for selecting an appropriate

promise index is that such an index function should agree with $f(\cdot)$ on regions in maximum depth, i.e., the regions that contain only one sample point. Moreover, the index function can also be constructed using ideas related to successful heuristics from branch-and-bound that combine (with appropriately constructed weights) data from an LP bound associated with the region with values obtained by sampling. This may be thought of as a structure-exploiting alternative to branch-and-bound heuristic estimation procedures based on integerization penalties (these compute estimates at a node by combining the LP bound at a node with so-called penalty pseudo-costs on non-integer-valued variables).

The NP method has a number of additional features worth highlighting here. The Markov property of the NP method provides a powerful approach for the study of the convergence rates and stopping rules. This research topic is closely connected to an exciting area of modern mathematical research, the study of quantitative convergence rates of Markov chains.

Another remarkable feature of the NP framework is that it can combine features of global search partitioning and local search heuristics in a natural way. It replaces the initialization or guessing procedures for finding initial solutions typical of local search methods by starting with a partition of the feasible region into an initial promising region (in which only a subset of the variables are fixed) and a complementary region. The NP algorithm can also take maximum advantage of parallel computer architectures. In every iteration, the algorithm looks at M + 1 regions, each of which can be handled independently in a parallel or distributed computing environment.

In order to take advantage of the supply chain problem structure via NP, we first construct a heuristic as described below to develop a set of efficiency indices for the warehouses. These indices provide a ranking of warehouses that is then used as a basis for the selection of a subset of the warehouses as a *"warm start"* promising region for NP. (This is not an initial solution in the sense that all warehouse variables are fixed, but rather a partial solution in which only some of them are fixed. Moreover, the complementary region contains all other possible solutions.) The ranking is then also used as a basis for biased sampling approaches for an NP framework that employs looping constructs available in the AMPL modeling language (these are also available in other modeling languages as well) in conjunction with the CPLEX MIP solver to generate and then solve subproblems corresponding to samples for the promising region and complementary region.

4.3.1 A Heuristic Procedure for Ranking Warehouses (LP-based)

We generate via the following three-step procedure a heuristic warehouse ranking based on average unit costs for the warehouses by solving (via CPLEX) a total cost optimization problem for each warehouse and then scaling the optimal value by dividing by the warehouse capacity:

Warehouse Ranking

1. Pick a warehouse from the set of warehouses and open it. Close all other warehouses. Solve the problem that requires that the total warehouse throughput equal the warehouse capacity, replacing the customers' "demand" equations by upper bounds that limit product-customer shipments to

at most the demand. (The integer variables corresponding to product-customer pairs are relaxed to continuous variables.) The optimal value of this linear programming problem (involving only one warehouse but all plants and customers) yields the lowest total cost associated with full utilization of the warehouse. (Note that this optimal value also includes the fixed cost of the warehouse.)

2. Calculate an average unit cost for each warehouse by the dividing the total cost by the (fully utilized) capacity of the warehouse.

3. Define an efficiency ranking for the set of warehouses according to the average unit costs. (We are currently experimenting with variants of this ranking process in which the throughput is set to an appropriate fraction of the warehouse capacity.)

4.3.2 An NP/CPLEX Implementation

We now describe an NP/CPLEX implementation that uses these unit cost rankings to generate a "warm start" followed by biased sampling guided by the global NP viewpoint. This approach involves the following parameters and sets:

- R: the rank cutoff value used for purposes of biased sampling (The top R ranked warehouses are placed in a set called TOP, and the remaining warehouses are placed into a set called $BOTTOM$.)
- p: the probability of choosing a warehouse from TOP in the sampling process
- $initNP$: the number of warehouses to be set as open at the start of each major iteration (initial promising region for the major iteration) (Here a major iteration is defined as the branching process from the starting point to the first node at which all warehouses are fixed (at which point a backtrack step is made to a new starting region).)
- S: the number of samples to be generated in each region (including the complementary regions) (Each sample is a set of W open warehouses in the feasible set of the corresponding region, and the optimal cost of the supply chain network with this set of warehouses is determined by CPLEX. (In this context, each sample requires about 3 seconds to evaluate on a 500MHz computer.))

NP/CPLEX Implementation for Supply Chain Design

The NP/CPLEX implementation is as follows (see Figure 2) For illustrative purposes, the figure assumes that an initial promising region is constructed by opening the 6 top-ranked warehouses:

1. If iteration = 1 then the initial promising region is obtained by fixing the highest ranked $initNP$ warehouses as open. Else obtain a promising region by fixing an appropriate subset of $BestOPEN$ (see below) as open.

2. Generate S samples for each of the following regions:

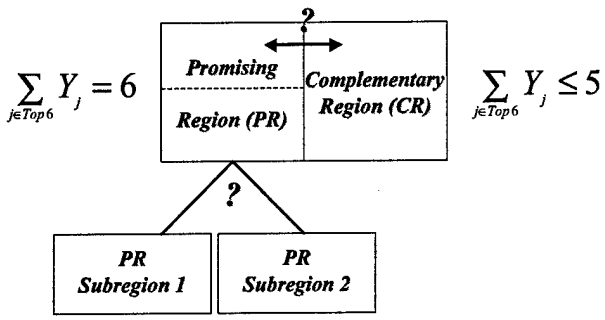

Figure 2: CPLEX/Nested Partitions/CPLEX Hybrid (C-NP-C)

- Region 1 (first subset of the promising region): another warehouse is selected and set as open (this warehouse is randomly selected from the set TOP with probability p, and from the set $BOTTOM$ with probability $(1-p)$)
- Region 2 (remainder of the promising region): the newly selected warehouse is closed
- Region 3: complementary region (at least one of the warehouses used to define the promising region is not allowed in the warehouse sets considered in this region)

For each region the remaining warehouses needed to complete each sample of W warehouses are chosen from TOP and $BOTTOM$, where the probability of picking a warehouse from TOP is p and picking a warehouse from $BOTTOM$ is $(1-p)$. CPLEX is used to find the optimal shipping pattern for each sample (set of warehouses). NP keeps track of the best sample as $BestOPEN$ and uses this information to either continue the path down from the promising region or to backtrack to a subset of the complementary region.

The remainder of the process is analogous to the generic NP procedure. If backtracking is needed because the bottom of the tree is reached (all warehouses are fixed) or because the best sample occurs in the surrounding region, then we choose $initNP$ warehouses from $BestOPEN$ as the new starting set (performing the selection so that this set does not coincide with a previous starting set) and return to step 1. In summary, NP utilizes the results of the warehouse ranking heuristic to begin a search process that initially seeks to focus on a warehouse subset predicted to be in the optimal solution according to this ranking, but the framework nevertheless maintains a global view of the feasible set and therefore allows completely different solutions (unrelated to the ranking) to be considered.

We note the similarity of this NP implementation to the heuristic portion of the new LR method discussed in section 4.2. In both methods, the focus is on the selection of good subsets of warehouses, which are then input to an MIP solver for evaluation. In our LR approach, a warehouse ranking takes place using information from Lagrangian multipliers; in NP, warehouse subsets are selected

	plants	warehouses	open warehouses (W)	customers	products
minimum	5	30	10	50	3
maximum	10	100	20	200	10

Table 2: Max and min values of design parameters

using information from single warehouse subproblems based on original costs, results from promise index considerations, and the biased sampling procedure.

Since our NP framework utilizes the solution of MIPs over partitions of the original feasible set, a lower bound could be obtained by choosing the smallest LP relaxation value over any such partition. While this lower bound would not be worse than that obtained by the LP relaxation of the standard MIP model (which is used in C-NP-C), in our computational experience as discussed below, better lower bounds can be obtained from the strong MIP model (either from its LP relaxation or from the new LR approach). Thus, in order to have a relatively good lower bound in order to provide a confidence interval for the C-NP-C results, the LP relaxation value of the strong MIP model is used as the lower bound. Note that the computation of this lower bound utilizes the same AMPL model used in the C-NP-C process, except that the cut constraints (4.1) are added. We are currently also pursuing research on other methods for lower bound computation based on duals generated during the C-NP-C process, and these alternatives are described in section 5.

5 Numerical Results

Using a factorial design approach, we set min and max values (see Table 2) for the 5 design parameters: numbers of plants, warehouses, open warehouses, customers, and products. This resulted in $2^5 = 32$ problem sets of different sizes. Since some of the smaller problems of this set were easily solved by the direct application of CPLEX, we subsequently generated ten additional larger problems by selectively increasing in some instances the number of customers to 250 or the number of products to 15 (these larger problems are numbered 33-42). See Table 3 for the full problem sizes of the smallest and the largest data sets generated. After randomly generating the data for each problem, we froze the data so that various solution strategies could be compared for each problem. (See Table 4 for the complete test problems suite.)

5.1 MIP CPLEX 8.1 vs Strong Formulation MIP CPLEX 8.1

Table 6 below summarizes the results of the 2-hour CPLEX 8.1 runs using the MIP formulations presented earlier. The strong formulation often provides superior lower bounds for the large problems. However, the feasible solutions obtained are often inferior, perhaps due to the fact that the

#	plants	warehouses	open warehouses	customers	products
1	5	30	10	50	3
	Problem Size:	constraints =	286		
		binary variables =	4,530		
		continuous variables =	450		
42	10	100	20	250	15
	Problem Size:	constraints =	5,501		
		binary variables =	375,000		
		continuous variables =	15,000		

Table 3: Problem dimensions for smallest (# 1) and largest (# 42) test problems

root relaxation times are much longer, and not much time is left to branch to explore the feasible region for good solutions.

5.2 Overall comparison of numerical results

In our computational tests for both the conventional and new LR methods, we initialize the Lagrangian multipliers to be the optimal dual variables from the strong formulation LP relaxations. The computational results indicate that the new LR method generates slightly better lower bounds than the conventional one. Moreover, it usually generates better feasible solutions as well. In only one instance among forty–two is the gap obtained by the new method larger than that obtained by the conventional, and it is sometimes significantly smaller.

The lower bounds provided by both Lagrangian methods provide slightly stronger bounds than those of the LP relaxation of the strong formulation (Table 5). This suggests that if we could maximize (or approximately maximize) the Lagrangian function efficiently without first solving the strong formulation LP relaxation, we could generate lower bounds that are significantly better than this relaxation. We have investigated approaches that use dual variables from the optimization problems associated with the NP hybrid approach, and believe that research in this direction may be promising since C-NP-C quickly provides high-quality feasible solutions and corresponding dual variables (the challenge here is that C-NP-C fixes many or all of the binary variables associated with the warehouses, so that only a partial set of duals is available).

Our computational experience (see Table 8) also demonstrates that while our new LR approach often yields significantly better feasible solutions than standalone CPLEX for the more difficult problems in the test suite, the use of CPLEX within the NP framework of the C-NP-C hybrid generally outperforms even the LR methods in terms of quality of feasible solutions. Specifically, for the 20 harder problems in the test suite, C-NP-C obtains the best solution in 15 cases, the new LR approach is better in 3 cases, while standalone CPLEX is better in only 2 cases (one of these cases corresponds to a standard MIP model, and the other is the stronger model). Moreover,

Problem #	*plants*	*warehouses*	*W fixed # warehouses*	*customers*	*products*
1	5	30	10	50	3
2	5	30	10	50	10
3	5	30	10	200	3
4	5	30	10	200	10
5	5	30	20	50	3
6	5	30	20	50	10
7	5	30	20	200	3
8	5	30	20	200	10
9	5	100	10	50	3
10	5	100	10	50	10
11	5	100	10	200	3
12	5	100	10	200	10
13	5	100	20	50	3
14	5	100	20	50	10
15	5	100	20	200	3
16	5	100	20	200	10
17	10	30	10	50	3
18	10	30	10	50	10
19	10	30	10	200	3
20	10	30	10	200	10
21	10	30	20	50	3
22	10	30	20	50	10
23	10	30	20	200	3
24	10	30	20	200	10
25	10	100	10	50	3
26	10	100	10	50	10
27	10	100	10	200	3
28	10	100	10	200	10
29	10	100	20	50	3
30	10	100	20	50	10
31	10	100	20	200	3
32	10	100	20	200	10
33	5	100	10	50	15
34	5	100	10	250	5
35	5	100	10	250	10
36	5	100	20	250	10
37	10	30	10	250	10
38	10	100	10	50	15
39	10	100	10	250	5
40	10	100	10	250	15
41	10	100	20	50	15
42	10	100	20	250	15

Table 4: Test problems suite

#	conventional	Time (sec)	strong	Time (sec)
1	312,344	0.13	359,898	2
2	1,011,794	0.56	1,291,474	31
3	1,314,169	0.48	1,571,885	12
4	3,958,458	2.19	5,145,410	240
5	312,344	0.11	328,332	2
6	1,011,794	0.52.	1,095,773	15
7	1,314,169	0.48	1,382,042	6
8	3,958,458	2.15	4,355,656	96
9	216,026	0.63	303,180	22
10	738,201	2.69	1,154,925	478
11	868,749	2.72	1,248,975	178
12	2,940,204	11.42	4,658,537	2736
13	216,026	0.64	249,952	14
14	738,201	2.62	945,729	390
15	868,749	2.57	1,047,292	137
16	2,940,204	11.05	3,851,205	2840
17	249,024	0.15	310,938	2
18	867,003	0.56	1,103,949	19
19	1,054,227	0.56	1,319,928	31
20	3,471,137	2.56	4,475,599	325
21	249,024	0.13	269,124	1
22	867,003	0.54	945,193	11
23	1,054,227	0.49	1,142,485	11
24	3,471,137	2.36	3,792,974	139
25	211,562	0.71	299,476	26
26	642,890	2.78	976,206	388
27	761,391	2.91	1,124,278	208
28	2,666,086	13.18	4,066,798	2787
29	211,562	0.67	251,446	16
30	642,890	2.67	803,064	254
31	761,391	2.97	924,693	124
32	2,666,086	12.27	3,374,194	1722
33	1,099,480	4.11	1,766,749	1284
34	1,812,550	6.32	2,797,910	981
35	3,633,169	13.89	5,753,405	3379
36	3,633,169	13.23	4,751,316	2453
37	4,379,097	3.27	5,674,061	404
38	988,572	4.99	1,536,800	1074
39	1,635,913	6.45	2,460,995	741
40	4,811,311	26.03	7,565,944	9438
41	988,572	4.66	1,259,151	872
42	4,811,311	24.62	6,217,881	4788

Table 5: Values of LP Relaxations (conventional vs. strong formulation)

#	LB	conventional UB	Gap(%)	LB	strong UB	Gap(%)
1	364,551	364,588	93 OP	364,551	364,588	5909 OP
2	1,338,657	1,339,077	0.03	1,335,991	1,341,108	0.38
3	1,587,341	1,587,500	758 OP	1,587,264	1,587,538	0.02
4	5,142,840	5,421,679	5.14	5,146,147	5,404,865	4.79
5	336,411	336,445	25 OP	336,411	336,445	114 OP
6	1,097,903	1,098,013	1013 OP	1,097,876	1,098,062	0.02
7	1,385,342	1,385,480	770 OP	1,385,357	1,385,480	3549 OP
8	4,381,708	4,382,040	2805 OP	4,372,600	4,382,335	0.22
9	312,051	312,088	863 OP	312,957	312,088	2109 OP
10	1,154,488	1,260,763	8.43	1,160,255	1,298,631	10.66
11	1,240,612	1,356,147	8.52	1,254,834	1,328,182	5.52
12	4,162,943	5,905,827	29.51	4,658,925	6,823,737	31.72
13	253,210	253,235	1610 OP	252,892	253,588	0.27
14	947,388	994,262	4.71	948,771	989,010	4.07
15	1,047,250	1,072,665	2.37	1,051,831	1,061,549	0.92
16	3,849,749	4,092,693	5.94	3,851,556	-	**NFSTL**
17	325,076	325,109	170 OP	325,074	325,109	447 OP
18	1,157,286	1,157,402	7163 OP	1,132,906	1,159,687	2.31
19	1,367,560	1,369,319	0.13	1,342,308	1,381,765	2.86
20	4,473,427	4,896,226	8.64	4,505,268	4,816,653	6.46
21	272,630	272,658	54 OP	272,630	272,658	317 OP
22	948,161	948,256	3321 OP	947,762	948,526	0.08
23	1,151,626	1,152,388	0.07	1,151,405	1,152,518	0.10
24	3,815,316	3,822,137	0.18	3,811,164	3,820,765	0.25
25	309,347	322,451	4.06	304,565	320,523	4.98
26	975,571	1,164,312	16.21	980,481	1,155,400	15.14
27	1,101,380	1,233,919	10.74	1,124,522	1,328,515	15.35
28	3,580,816	5,263,489	31.97	4,067,227	5,528,464	26.43
29	256,561	256,588	5531 OP	254,391	259,860	2.10
30	803,005	860,173	6.65	803,350	862,556	6.86
31	924,811	950,509	2.70	927,633	942,338	1.56
32	3,372,835	3,688,002	8.55	3,374,311	3,769,076	10.47
33	1,765,916	2,008,789	12.09	1,767,087	2,112,366	16.35
34	2,618,971	3,357,485	22.00	2,797,910	**NFSTL**	-
35	4,745,800	7,521,489	36.90	5,753,880	8,428,109	31.73
36	4,659,669	5,110,524	8.82	4,751,418	5,353,491	11.25
37	5,671,983	6,479,411	12.46	5,714,385	6,651,057	14.08
38	1,535,728	1,878,195	18.23	1,537,089	1,909,403	19.50
39	2,303,216	3,009,080	23.46	2,461,300	3,346,689	26.46
40	5,639,197	9,759,251	42.22	7,565,944 [1]	**NFSTL**	-
41	1,258,827	1,351,726	6.87	1,259,284	1,344,238	6.32
42	5,440,735	8,019,040	32.15	6,218,055	**NFSTL**	-

Table 6: Comparison of 2-hour CPLEX 8.1 runs (conventional vs. strong formulation). **X OP** indicates optimal solution obtained in X seconds; **NFSTL** indicates "no feasible solution obtained in time limit

because of the effectiveness of the fast warm starting procedure employed by NP, the times required by C-NP-C to reach those highest quality solutions are generally only a few minutes (after which only rarely is a slightly better solution found). This is in strong contrast to all of the other methods which either typically produce poor quality solutions even after two hours (standard MIP approach) or which require initial solution of the relaxation of the strong MIP formulation, an initialization procedure that itself often requires an hour or more (and, in one case, more than two hours) for the more difficult problems. Thus, the NP framework for the use of BC solvers has proven to be efficient, reliably fast, and effective in terms of providing high-quality solutions.

#	LB	conventional UB	*Gap (%)*	LB	new UB	*Gap (%)*
1	359,898	387,058	7.02	361,703	364,732	0.83
2	1,291,474	1,421,743	9.16	1,297,676	1,341,939	3.30
3	1,571,885	1,788,259	12.10	1,572,982	1,588,095	0.95
4	5,145,410	5,620,029	8.45	5,148,021	5,392,345	4.53
5	328,338	346,995	5.38	330,004	337,763	2.30
6	1,095,773	1,109,605	1.25	1,097,500	1,105,361	0.71
7	1,382,050	1,399,606	1.25	1,385,150	1,387,668	0.18
8	4,355,656	4,429,624	1.67	4,360,140	4,382,279	0.51
9	303,180	431,800	29.79	304,590	319,109	4.55
10	1,154,925	1,456,219	20.69	1,159,971	1,267,463	8.48
11	1,248,975	1,514,917	17.56	1,250,556	1,318,355	5.14
12	4,658,537	5,563,920	16.27	4,658,912	5,240,857	11.10
13	249,952	321,686	22.30	250,980	253,354	0.94
14	945,729	1,086,835	12.98	947,015	999,058	5.21
15	1,047,292	1,246,990	16.01	1,049,092	1,066,242	1.61
16	3,851,205	4,266,963	9.74	3,851,423	4,063,299	5.21
17	310,938	365,599	14.95	312,395	326,798	4.41
18	1,103,950	1,189,556	7.20	1,109,058	1,158,106	4.24
19	1,319,930	1,418,915	6.98	1,324,283	1,375,694	3.74
20	4,475,600	4,885,294	8.39	4,481,829	4,853,653	7.66
21	269,129	277,044	2.86	270,380	272,905	0.93
22	945,198	966,647	2.22	947,934	952,218	0.45
23	1,142,541	1,166,517	2.06	1,144,638	1,154,686	0.87
24	3,792,974	3,834,267	1.08	3,798,546	3,823,743	0.66
25	299,476	400,508	25.23	300,839	324,375	7.26
26	976,206	1,199,867	18.64	980,089	1,126,702	13.01
27	1,124,280	1,401,298	19.77	1,126,600	1,238,026	9.00
28	4,066,798	4,979,825	18.33	4,067,756	4,780,657	14.91
29	251,446	301,796	16.68	252,397	260,898	3.26
30	803,064	917,686	12.49	806,022	858,294	6.09
31	924,693	1,024,008	9.70	926,208	947,068	2.20
32	3,374,194	3,737,265	9.72	3,375,278	3,589,712	5.97
33	1,766,750	2,161,456	18.26	1,767,329	2,007,048	11.94
34	2,797,910	1,323,887	15.82	2,798,405	3,139,575	10.87
35	5,753,405	6,490,988	11.36	5,753,470	6,476,127	11.16
36	4,751,316	5,142,009	7.60	4,751,486	5,042,011	5.76
37	5,674,061	6,490,148	12.57	5,674,755	6,281,830	9.66
38	1,536,800	1,917,258	19.84	1,539,309	1,793,900	14.19
39	2,460,995	2,956,774	16.77	2,461,245	2,705,557	9.03
40	7,565,944	8,924,669	15.22	7,566,983	8,850,466	14.50
41	1,259,151	1,423,115	11.52	1,262,273	1,358,110	7.06
42	6,217,881	6,630,528	6.22	6,219,078	6,660,339	6.63

Table 7: Comparison of Lagrangian Relaxation runs using strong LP duals (conventional vs. new formulation)

#	MIP LB	MIP UB	Gap (%)	strong MIP LB	strong MIP UB	Gap (%)	conv. LG LB	conv. LG UB	Gap (%)	new LG LB	new LG UB	Gap (%)	C-NP-C LB	C-NP-C UB	Gap (%)
10	1,154,488	1,260,763	8.43	1,160,255	1,298,631	10.66	1,154,925	1,456,219	20.69	1,159,971	1,267,463	8.48	1,154,925	**1,255,900**	8.04
11	1,240,612	1,356,147	8.52	1,254,834	1,328,182	5.52	1,248,975	1,514,917	17.56	1,250,556	**1,318,355**	5.14	1,248,975	1,322,780	5.58
12	4,162,943	5,905,827	29.51	4,658,925	6,823,737	31.72	4,658,537	5,563,920	16.27	4,658,912	5,240,857	11.10	4,658,537	**5,024,820**	7.29
16	3,849,749	4,092,693	5.94	3,851,556	NFSTL	-	3,851,205	4,266,963	9.74	3,851,423	4,063,299	5.21	3,851,205	**4,033,560**	4.52
20	4,473,427	4,896,226	8.64	4,505,268	4,816,653	6.46	4,475,600	4,885,294	8.39	4,481,829	4,853,653	7.66	4,475,599	**4,812,650**	7.00
26	975,571	1,164,312	16.21	980,481	1,155,400	15.14	976,206	1,199,867	18.64	980,089	1,126,702	13.01	976,206	**1,104,190**	11.59
27	1,101,380	**1,233,919**	10.74	1,124,522	1,328,515	15.35	1,124,280	1,401,298	19.77	1,126,600	1,238,026	9.00	1,124,278	1,235,210	8.98
28	3,580,816	5,263,489	31.97	4,067,227	5,528,464	26.43	4,066,798	4,979,825	18.33	4,067,756	4,780,657	14.91	4,066,798	**4,545,020**	10.52
30	803,005	860,173	6.65	803,350	862,556	6.86	803,064	917,686	12.49	806,022	858,294	6.09	803,064	**852,053**	5.75
32	3,372,835	3,688,002	8.55	3,374,311	3,769,076	10.47	3,374,194	3,737,265	9.72	3,375,278	3,589,712	5.97	3,374,194	**3,514,680**	4.00
33	1,765,916	2,008,789	12.09	1,767,087	2,112,366	16.35	1,766,750	2,161,456	18.26	1,767,329	2,007,048	11.94	1,766,749	**1,992,120**	11.31
34	2,618,971	3,357,485	22.00	2,797,910	NFSTL	-	2,797,910	1,323,887	15.82	2,798,405	**3,139,575**	10.87	2,797,910	3,141,820	10.95
35	4,745,800	7,521,489	36.90	5,753,880	8,428,109	31.73	5,753,405	6,490,988	11.36	5,753,470	6,476,127	11.16	5,753,405	**6,321,140**	8.98
36	4,659,669	5,110,524	8.82	4,751,418	5,353,491	11.25	4,751,316	5,142,009	7.60	4,751,486	5,042,011	5.76	4,751,316	**5,021,480**	5.38
37	5,671,983	6,479,411	12.46	5,714,385	6,651,057	14.08	5,674,061	6,490,148	12.57	5,674,755	6,281,830	9.66	5,674,061	**6,254,060**	9.27
38	1,535,728	1,878,195	18.23	1,537,089	1,909,403	19.50	1,536,800	1,917,258	19.84	1,539,309	1,793,900	14.19	1,536,800	**1,775,690**	13.45
39	2,303,216	3,009,080	23.46	2,461,300	3,346,689	26.46	2,460,995	2,956,774	16.77	2,461,245	**2,705,557**	9.03	2,460,995	2,725,680	9.71
40[a]	5,639,197	9,759,251	42.22	7,565,944	NFSTL	-	7,565,944	8,924,669	15.22	7,566,983	8,850,466	14.50	7,565,944	**8,720,900**	13.24
41	1,258,827	1,351,726	6.87	1,259,284	**1,344,238**	6.32	1,259,151	1,423,115	11.52	1,262,273	1,358,110	7.06	1,259,151	1,357,000	7.21
42	5,440,735	8,019,040	32.15	6,218,055	NFSTL	-	6,217,881	6,630,528	6.22	6,219,078	6,660,339	6.63	6,217,881	**6,600,730**	5.80

Table 8: Comparison of 5 methods for 20 difficult problems (best feasible solution in boldface)

[a]Strong LP relaxation requires nearly 3 hours for problem # 40.

6 Conclusions and Directions for Future Research

Our computational results demonstrate that for large-scale supply chain design problems, the Nested Partitions approach significantly outperforms both general-purpose combinatorial optimizers (such the branch-and-cut solver within CPLEX) and specialized approaches such as those based on Lagrangian relaxation. The Nested Partitions framework can effectively combine problem–specific heuristics with MIP tools (such as AMPL/CPLEX), so for this problem class we have developed an excellent warehouse ranking heuristic and used this heuristic to construct a "warm start" procedure followed by an effective biased sampling approach (using CPLEX to solve the MIPs corresponding to the samples) that uses this ranking and is guided by a global view of the problem. This research has also established the applicability of the AMPL/CPLEX modeling-language/solver combination as an excellent means for the implementation of the NP framework for global meta-hybrids, and thus represents a novel and successful use of these powerful software tools.

There are many directions for further investigation suggested by our approach. One particularly promising direction that we are pursuing involves the construction of composite lists of preferred warehouses. In the results above, we used biased sampling that relied solely on an optimization-based initial ranking of the warehouse list. Rather than simply relying on one ranking determined at the start of the procedure, we have also considered more sophisticated approaches such as the construction of composite highly-ranked subsets comprised of warehouses selected via two different initial ranking schemes (for example, combining warehouses ranked highly by either efficiency or capacity has produced excellent results in some problems). Alternatively, the highly-ranked group could evolve dynamically by adding to it new warehouses that appear in high-quality solutions. For the larger instances we considered, no method appears to be capable of generating high-quality lower bounds quickly. Significant time is required for CPLEX to generate reasonable bounds, either by branching (conventional formulation) or by solving the root LP relaxation (strong formulation). Consequently, we are investigating approaches for the generation of lower bounds using information from the NP framework. For example, dual information from the samples generated within the NP framework could used or sample information could be coordinated via a column generation approach, and the corresponding duals from this approach could be used in a lower bound generation process. Finally, the promise index for a region could be made a function of both lower bound (LP relaxation for the region or a superset of the region) and upper bound (sampling) information, since both types of information are available within this hybrid approach.

The Multicommodity Distribution System Design (MDSD) problem class investigated in this research has a constraint, based upon interaction with our industrial partner (and also quite common in the literature) requiring the selection of exactly W warehouses from the list of candidate locations. An interesting and equally-important alternative is the problem class in which at most W warehouses can be chosen to satisfy the constraints. The Lagrangian Relaxation and hybrid

C-NP-C methods considered here may be suitably modified to handle that problem class.

Other problem areas well-suited to optimization-based implementation of the NP framework are also under investigation. For example, problems arising from radiation treatment planning optimization (D'Souza et al., 2003) as well as the design of large-scale internet data centers (Santos et al., 2002) appear particularly appropriate for our approach since they can be thought of involving "assignments of certain critical resources followed by optimization of the processes that depend on those assignments. Good problem-specific ranking processes for the critical resources are available for those problem classes. Using these ranking within the framework of NP-directed biased sampling (with samples being evaluated by using branch-and-cut to solve MIPs in the remaining variables) is expected to provide a highly-effective approach to these large and extremely challenging combinatorial problems. We have demonstrated in this paper that the looping constructs within well-known optimization modeling languages combined with good branch-and-cut solvers provide an excellent tool for implementing this approach to combinatorial optimization, and we plan to continue our development of such methods and software for a variety of challenging problem classes.

7 Acknowledgements

This work was supported by the National Science Foundation under grant DMI-0100220. The authors thank Winston Yang who made many useful contributions at an early stage of this research.

References

Aardal, K., 1998, Capacitated Facility Location: Separation Algorithms and Computational Experience, Mathematical Programming 81, 149–175

Beasley, J., 1993, Lagrangian Heuristics for Location Problems, European Journal of Operational Research 65, 383–399

Cordeau, J., Soumis, F., and Desrosiers, J., 2001, Simultaneous assignment of locomotives and cars to passenger trains, Operations Research 49, 531–548

Crainic, T., Toulouse, M., and Gendreau, M., 1996, Parallel asynchronous tabu search for multicommodity location-allocation with balancing requirements, Annals of Operations Research 63, 277–299

Delaney, R., 2003, 14th Annual State Of Logistics Report, June 2, National Press Club Washington, D. C.

Delmaire, H., Daz, J., and Fernndez, E., 1999, Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem, INFOR, Canadian Journal of Operational Research and Information Processing 37(3), 194–225

D'Souza, W., Meyer, R., Naqvi, S., and Shi, L., 2003, Beam orientation optimization in imrt using single beam characteristics and mixed-integer formulations, AAPM Annual Meeting, San Diego

Fisher, M., 1981, The Lagrangian Relaxation Method for Solving Integer Programming Problems, Management Sciences 27, 1–18

Garey, M. and Johnson, D., 1979, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco

Geoffrion, A. and Graves, G., 1974, Multicommodity Distribution System Design by Benders Decomposition, Management Science 20, 822–844

Hindi, K. and Basta, T., 1994, Computationally Efficient Solution of a Multiproduct, Two-Stage Distribution-Location Problem, Journal of the Operational Research Society 45, 1316–1323

Klincewicz, J. and Luss, H., 1986, A Lagrangian Relaxation Heuristic for Capacitated Facility Location with Single-Source Constraints, Journal of Operational Research Society 37, 495–500

Lee, C., 1993, A Cross Decomposition Algorithm for A Multiproduct-Multitype Facility Location Problem, Computers and Operations Research 20, 527–540

Mathar, R. and Niessen, T., 2000, Optimum positioning of base stations for cellular radio networks, Wireless Networks 6(6), 421–428

Mitchell, J., 2000, Branch-and-Cut Algorithms for Combinatorial Optimization Problems, to appear in the Handbook of Applied Optimization, Oxford University Press

Neebe, A. and Rao, M., 1983, An Algorithm for the Fixed-Charge Assigning Users to Sources Problem, Journal of Operational Research Society 34, 1107–1113

Pirkul, H. and Jayaraman, V., 1996, Production, Transportation, and Distribution Planning in a Multi-Commodity Tri-Echelon System, Transportation Sciences 30(4), 291–302

Santos, C., Zhu, X., and Crowder, H., 2002, A Mathematical Optimization Approach for Resource Allocation in Large Scale Data Centers, Technical Report HPL-2002-64(R.1), Intelligent Enterprise Technologies Laboratory HP Laboratories Palo Alto, http://www.hpl.hp.com/techreports/2002/HPL-2002-64R1.pdf

Shi, L. and Ólafsson, S., 2000, Nested Partitions Method for Global Optimization, Operations Research 48, 390–407

Shi, L., Ólafsson, S., and Chen, Q., 2001, An Optimization Framework for Product Design, Management Science 47(12), 1681–1692

Simchi-Levi, D. and Bramel, J., 1997, The Logic of Logistics: theory, algorithms, and applications for logistics management, Springer Series in Operations Research

Wolsey, L., 1998, Integer Programming, John Wiley & Sons, Inc.

Appendix B – AMPL Code for C-NP-C Hybrid

Appendix B – AMPL Code for C-NP-C Hybrid

```
reset data;

#NUMBER OF WAREHOUSES TO PICK FROM THE UNIT COST RANKINGS
param FixUnit default 20;

#NUMBER OF WAREHOUSES TO PICK FROM THE FIXED COST RANKINGS, DO NOT FORGET
#COMMENT OUT THE FOR LOOP BELOW IF YOU ARE NOT USING THESE RANKINGS
param FixFixed default 0;

#THE NUMBER OF WAREHOUSES TO FIX INITIALLY IN NP SCHEMA
param NP default 6;

#number of samples TO GENERATE IN EACH REGION
param S default 6;

#THE PROBABILITY OF CHOOSING A WAREHOUSE FROM THE BOTTOM WHEN SAMPLING
param prob;
let prob := 0;

#NUMBER OF ITERATIONS FOR THE NP RUN
param numIterations default 2000;

# ------------------------------------------------------------------------
param plants     default 5;
param warehouses default 100;
param W          default 10;
param customers  default 50;
param products   default 10;
# ------------------------------------------------------------------------

set K := 1..products;
set J := 1..warehouses;
set I := 0..(customers - 1);
set L := 1..plants;

printf "%-20s%3d\n", "plants", plants;
printf "%-20s%3d\n", "warehouses", warehouses;
printf "%-20s%3d\n", "open warehouses", W;
printf "%-20s%3d\n", "customers", customers;
printf "%-20s%3d\n", "products", products;
printf "\n";

# ------------------------------------------------------------------------
```

```
param c {L, J, K} >= 0; # shipping cost from plant l to warehouse j of pr
param d {J, I, K} >= 0; # shipping cost from warehouse j to customer i
                        # of product k
param f {J}       >= 0; # fixed cost of warehouse j
param q {J}       >= 0; # capacity of warehouse j
param s {K}       >= 0; # volume of one unit of product k
param v {L, K}    >= 0; # supply at plant l of product k
param w {I, K}    >= 0; # demand of customer i for product k
# ------------------------------------------------------------------------

option randseed '7777';
param temp;
param avg_c;
param avg_d;
param totalUnits;

#printf "Randomly generating c...\n";
for {(j, k, l) in {J, K, L}}
  let c[l, j, k] := Uniform(0,200);

#printf "Randomly generating d...\n";
for {(i, j, k) in {I, J, K}}
  let d[j, i, k] := Uniform(0,200);

#printf "Randomly generating s...\n";
for {k in K}
  let s[k] := Uniform(10,20);

#printf "Randomly generating w...\n";
for {(i, k) in {I, K}}
  let w[i, k] := Uniform(10,99);

#printf "Randomly generating v...\n";
for {(k, l) in {K, L}} {
  let temp := (sum {i in I} w[i, k]) / plants;
  let v[l, k] := Uniform(temp, 2.5 * temp);
}

#printf "Randomly generating q...\n";
for {j in J} {
  let temp := (sum{(i, k) in {I, K}} w[i, k] * s[k]) / W;
  let q[j] := Uniform(0.95 * temp, 1.33 * temp);
}

let avg_c      := (sum {(l, j, k) in {L, J, K}} c[l, j, k]) / card({L, J,
```

```
let avg_d      := (sum {(j, i, k) in {J, I, K}} d[j, i, k]) / card({J, I,
let totalUnits := sum{(i, k) in {I, K}} w[i, k];
let temp       := (avg_c + avg_d) * totalUnits / 18 / W;

#printf "Randomly generating f...\n";
for {j in J}
  let f[j] := Uniform(temp, 2 * temp);

#-------------------------------------------------------------------------
var U {L, J, K} >= 0;   # units from plant l to warehouse j of product k
var X {J, I, K} binary; # 1 iff warehouse j supplies customer i with prod
var Y {J}       binary; # 1 iff warehouse j is open
var dummyX {J, I, K} >= 0;

subject to constraintWarehouseCapacity2 {j in J}:

  sum {(i, k) in {I, K}} w[i, k] * s[k] * dummyX[j, i, k] >= q[j] * Y[j];

subject to constraintCustomerOneWarehousePerProduct2 {(i, k) in {I, K}}:

  sum {j in J} dummyX[j, i, k] <= 1;

subject to constraintWarehouseInflowEqualsOutflow2 {(j, k) in {J, K}}:

  sum {l in L} U[l, j, k] = sum {i in I} w[i, k] * dummyX[j, i, k];

subject to constraintWarehouseCapacity3 {j in J}:

  sum {(i, k) in {I, K}} w[i, k] * s[k] * dummyX[j, i, k] <= q[j] * Y[j];

subject to constraintCustomerOneWarehousePerProduct3 {(i, k) in {I, K}}:

  sum {j in J} dummyX[j, i, k] = 1;

# -------------------------------------------------------------------------
# shipments to customers cannot exceed warehouse capacity

subject to constraintWarehouseCapacity {j in J}:

  sum {(i, k) in {I, K}} w[i, k] * s[k] * X[j, i, k] <= q[j] * Y[j];

# -------------------------------------------------------------------------
# open W warehouses

subject to constraintOpenWarehouses:
```

```
  sum {j in J} Y[j] = W;
# ------------------------------------------------------------------------
# a given customer can receive a given product from exactly 1 warehouse

subject to constraintCustomerOneWarehousePerProduct {(i, k) in {I, K}}:

  sum {j in J} X[j, i, k] = 1;

# ------------------------------------------------------------------------
# shipments to warehouses cannot exceed plant capacity

subject to constraintPlantCapacity {(k, l) in {K, L}}:

  sum {j in J} U[l, j, k] <= v[l, k];

# ------------------------------------------------------------------------
# what comes into a warehouse must go out

subject to constraintWarehouseInflowEqualsOutflow {(j, k) in {J, K}}:

  sum {l in L} U[l, j, k] = sum {i in I} w[i, k] * X[j, i, k];

# ------------------------------------------------------------------------
param theta {J, K};
param lambda {I, K};
param tempMax;
param tempMin;
param subprob default 1;
param subprobCost {J};


#for {(j, k) in {J, K}}
#  let theta[j, k] := 1 + min {l in L} c[l, j, k];

#for {(i, k) in {I, K}} {
#  let tempMax      := max {j in J} w[i, k] * (theta[j, k] + d[j, i, k]);
#  let tempMin      := min {j in J} w[i, k] * (theta[j, k] + d[j, i, k]);
#  let lambda[i, k] := 0.45 * tempMax + 0.55 * tempMin;
#}

# ------------------------------------------------------------------------
minimize totalCostLR1:

  sum {(j, k, l) in {J, K, L}}    (c[l, j, k] - theta[j, k]) * U[l, j, k]
```

```
# ------------------------------------------------------------------------
minimize totalCostLR2:

  sum {j in J}                          f[j] * Y[j]

  + sum {(i, j, k) in {I, J, K}}    (d[j, i, k] * w[i, k] - lambda[i, k]

                                    + theta[j, k] * w[i, k]) * X[j, i, k]
# ------------------------------------------------------------------------
minimize totalCostLR2Sub:

  f[subprob] * Y[subprob] +

  sum {(i, k) in {I, K}}     (d[subprob, i, k] * w[i, k] - lambda[i, k]

                            + theta[subprob, k] * w[i, k]) * X[subprob, i
# ------------------------------------------------------------------------
minimize cost:

  sum {j in J}                          f[j] * Y[j]
  + sum {(j, k, l) in {J, K, L}}    c[l, j, k] * U[l, j, k]

  + sum {(i, j, k) in {I, J, K}}    d[j, i, k] * w[i, k] * X[j, i, k];
# ------------------------------------------------------------------------
minimize cost2:

    sum {j in J}                        f[j] * Y[j]

  + sum {(j, k, l) in {J, K, L}}    c[l, j, k] * U[l, j, k]

  + sum {(i, j, k) in {I, J, K}}    d[j, i, k] * w[i, k] * dummyX[j, i, k
#------------------------------------------------------------------------

set WAREHOUSES;
set ARG_MIN;
set OPEN;
set BEST_OPEN;
set PUT_IN;
set PUT_IN_ONE;
set PUT_IN_ZERO;

param errorTheta {J, K};
param errorLambda {I, K};
param thetaRho;
param lambdaRho;
```

```
param lowerBound;
param bestLowerBound default -Infinity;
param bestUpperBound default Infinity;

param iterations default 0;
param gap default 1;
param feasible_count default 0;

param value default 1;

param SortedWarehouses {J};
param SortedWarehousesFixed {J};

param costwh {J};
data dual10rank;

drop constraintWarehouseCapacity3;
drop constraintWarehouseCapacity2;
drop constraintWarehouseCapacity;
drop constraintOpenWarehouses;
drop constraintCustomerOneWarehousePerProduct3;
drop constraintCustomerOneWarehousePerProduct;
drop constraintWarehouseInflowEqualsOutflow;
objective cost2;

#for {j in J} {
# for {k in J} fix Y[k] := 0;
 #       restore constraintWarehouseCapacity2[j];
  #      fix Y[j] := 1;
  #      solve;
  #      let costwh[j] := cost2/q[j];
  #      unfix Y;
  #      drop constraintWarehouseCapacity2[j];
#}

drop constraintWarehouseInflowEqualsOutflow2;
drop constraintCustomerOneWarehousePerProduct2;
restore constraintWarehouseCapacity;
restore constraintOpenWarehouses;
restore constraintCustomerOneWarehousePerProduct;
restore constraintWarehouseInflowEqualsOutflow;
objective cost;

let WAREHOUSES := J;
```

```
let value := 1;

for {j in J} {
  let tempMin := min {ww in WAREHOUSES} costwh[ww];
  for {ww in WAREHOUSES: costwh[ww] = tempMin} {
    let SortedWarehouses[value] := ww;
    let WAREHOUSES := WAREHOUSES diff {ww};
    let value := value + 1;
    break;
  }
  if value = warehouses + 1 then break;
}

let WAREHOUSES := J;
let value := 1;
for {j in J} {
  let tempMin := min {ww in WAREHOUSES} f[ww];
  for {ww in WAREHOUSES: f[ww] = tempMin} {
    let SortedWarehousesFixed[value] := ww;
    let WAREHOUSES := WAREHOUSES diff {ww};
    let value := value + 1;
    break;
  }
  if value = warehouses + 1 then break;
}

set TOP;
set BOTTOM;

let TOP := {};
let BOTTOM := {};

for {j in 1..FixUnit} {
let TOP := TOP union {SortedWarehouses[j]};
}

# THIS FOR LOOP HAS TO BE COMMENTED OUT FOR THE CASE WHERE WE ARE
# JUST USING THE UNIT COST RANKINGS
#for {j in 1..FixFixed} {
#        let TOP := TOP union {SortedWarehousesFixed[j]};
#}

let BOTTOM := J diff TOP;

param counter default 0;
```

```
param SortedMixTop {1..card(TOP)};
param SortedMixBottom {1..card(BOTTOM)};

let counter := 1;
for {j in TOP} {
        let SortedMixTop[counter] := j;
        let counter := counter + 1;
}

let counter := 1;
for {j in BOTTOM} {
        let SortedMixBottom[counter] := j;
        let counter := counter + 1;
}

param bestPromisingBound;
#let bestPromisingBound := bestUpperBound;
let bestPromisingBound := Infinity;
set Pro_OPEN;
let Pro_OPEN := {};

#let Pro_OPEN := BEST_OPEN;
#for {j in 1..W} let Pro_OPEN := Pro_OPEN union {SortedWarehouses[j]};

let value := 1;
let PUT_IN := {};
let PUT_IN_ONE := {};
let PUT_IN_ZERO := {};

param count default 0;
unfix Y;

let iterations := 0;

param NPdefault;
let NPdefault := NP;

option solver_msg 1;
option omit_zero_rows 1;
  option cplex_options 'mipgap = 0.03'
  'timelimit = 10'
  'timing = 1'
  'solutionlim = 1';
```

```
  restore constraintOpenWarehouses;
  restore constraintWarehouseCapacity;
  restore constraintCustomerOneWarehousePerProduct;
  restore constraintPlantCapacity;
  restore constraintWarehouseInflowEqualsOutflow;
  objective cost;

param SurroundValue;
param PromisingValue;
param mark default 0;
param CheckMe;
param key;

let CheckMe := 0;

set copyTOP;
set copyBOTTOM;

let copyTOP := TOP;
let copyBOTTOM := BOTTOM;

set SurroundTOP;
set SurroundBOTTOM;

let SurroundTOP := TOP;
let SurroundBOTTOM := BOTTOM;

param randomizer;

let counter := 0;

#repeat loop1 {

# if CheckMe = 0 then let randomizer := Uniform01();

 #       if (randomizer <= prob) then {
# let key := SortedMixBottom[(Irand224() mod card(BOTTOM))+1];
# let CheckMe := 1;
# if (key in copyBOTTOM) then {
 #        let PUT_IN_ONE := PUT_IN_ONE union {key};
  #               let counter := counter + 1;
# let copyBOTTOM := copyBOTTOM diff {key};
# let CheckMe := 0;
# }
# }
```

```
 #       if (randomizer > prob) then {
# let key := SortedMixTop[(Irand224() mod card(TOP))+1];
# let CheckMe := 1;
# if (key in copyTOP) then {
 #        let PUT_IN_ONE := PUT_IN_ONE union {key};
  #               let counter := counter + 1;
# let copyTOP := copyTOP diff {key};
# let CheckMe := 0;
# }
 #       }
  #      #display CheckMe;
   #     if counter = NP then break loop1;
#}



for {j in TOP} {
let PUT_IN_ONE := PUT_IN_ONE union {j};
let copyTOP := copyTOP diff {j};
let counter := counter + 1;
if counter = NP then break;
}

display PUT_IN_ONE;

set CcopyBOTTOM;
set CcopyTOP;

let CcopyBOTTOM := copyBOTTOM;
let CcopyTOP := copyTOP;

param SpecialKey;

param keyTOP;
param keyBOTTOM;

shell 'date';

repeat until (iterations = numIterations) {
let iterations := iterations + 1;
display iterations;

for {t in 1..S} {
shell 'date';
```

```
let counter := 0;
for {j in PUT_IN_ONE} {
fix Y[j]:= 1;
}
for {j in PUT_IN_ZERO} fix Y[j] := 0;

let CheckMe := 0;

repeat loop4 {
if counter = W-NP-1 then break loop4;
let randomizer := Uniform01();

if (randomizer <= prob) then
repeat until (CheckMe = 1) {
                        let key := SortedMixBottom[(Irand224() mod card(B
      if (key in copyBOTTOM) then {
fix Y[key] := 1;
                        let counter := counter + 1;
let copyBOTTOM := copyBOTTOM diff {key};
let CheckMe := 1;
}
}

if (randomizer > prob) then
repeat until (CheckMe = 1) {
                        let key := SortedMixTop[(Irand224() mod card(TOP)
                        if (key in copyTOP) then {
                        fix Y[key] := 1;
                        let counter := counter + 1;
                        let copyTOP := copyTOP diff {key};
                        let CheckMe := 1;
                    }
              }

if counter = W-NP-1 then break loop4;
let CheckMe := 0;
}

if t=1 then let randomizer := Uniform01();
else let randomizer := -1;

let CheckMe := 0;

if (randomizer <= prob and randomizer >= 0) then
       repeat until (CheckMe = 1) {
```

```
            let key := SortedMixBottom[(Irand224() mod card(BOTTOM))+1];
                 if (key in copyBOTTOM) then {
                  fix Y[key] := 1;
let SpecialKey := key;
let CcopyBOTTOM := CcopyBOTTOM diff {key};
                       let CheckMe := 1;
                }
         }

        if (randomizer > prob) then
         repeat until (CheckMe = 1) {
             let key := SortedMixTop[(Irand224() mod card(TOP))+1];
                  if (key in copyTOP) then {
                   fix Y[key] := 1;
                       let SpecialKey := key;
let CcopyTOP := CcopyTOP diff {key};
                       let CheckMe := 1;
                  }
            }

let copyBOTTOM := CcopyBOTTOM;
let copyTOP := CcopyTOP;

fix Y[SpecialKey] := 1;

solve;
if ((solve_result='solved' or solve_result_num=422)
and bestPromisingBound > cost) then {
let Pro_OPEN := {};
let bestPromisingBound := cost;
for {j in J:Y[j]=1} let Pro_OPEN := Pro_OPEN union {j};
let mark := 1;
}

display Y;
unfix Y;
}

for {t in 1..S} {
let counter := 0;

      for {j in PUT_IN_ONE} {
fix Y[j]:= 1;
}
for {j in PUT_IN_ZERO} fix Y[j] := 0;
```

```
let CheckMe := 0;

      fix Y[SpecialKey] := 0;

repeat loop9 {
let CheckMe := 0;
let randomizer := Uniform01();
if counter = W-NP then break loop9;

            if (randomizer <= prob) then
             repeat until (CheckMe = 1) {
                  let key := SortedMixBottom[(Irand224() mod card(BOTTOM
                       if (key in copyBOTTOM) then {
                        fix Y[key] := 1;
                             let counter := counter + 1;
                             let copyBOTTOM := copyBOTTOM diff {key};
                             let CheckMe := 1;
                       }
                 }

            if (randomizer > prob) then
             repeat until (CheckMe = 1) {
                  let key := SortedMixTop[(Irand224() mod card(TOP))+1];
                       if (key in copyTOP) then {
                        fix Y[key] := 1;
                             let counter := counter + 1;
                             let copyTOP := copyTOP diff {key};
                             let CheckMe := 1;
                        }
                 }

if counter = W-NP then break loop9;
}

solve;
if ((solve_result='solved' or solve_result_num=422)
and bestPromisingBound > cost) then {
       let Pro_OPEN := {};
            let bestPromisingBound := cost;
            for {j in J:Y[j]=1} let Pro_OPEN := Pro_OPEN union {j};
let mark := 2;
      }

let copyBOTTOM := CcopyBOTTOM;
```

```
      let copyTOP := CcopyTOP;
display Y;
      unfix Y;
}

for {t in 1..S} {
let counter := 0;
      let SurroundValue := (Irand224() mod NP) + 1;
for {j in PUT_IN_ONE} {
let counter := counter + 1;
            if (counter = SurroundValue) then {
             fix Y[j] := 0;
if (j in SurroundTOP) then let SurroundTOP := SurroundTOP diff {j};
else let SurroundBOTTOM := SurroundBOTTOM diff {j};
                  break;
}
      }

let counter := 0;
repeat loop12 {
let CheckMe := 0;
let randomizer := Uniform01();
            if (randomizer <= prob) then
                repeat until (CheckMe = 1) {
                  let key := SortedMixBottom[(Irand224() mod card(BOTTOM
                        if (key in SurroundBOTTOM) then {
                         fix Y[key] := 1;
                              let counter := counter + 1;
                              let SurroundBOTTOM := SurroundBOTTOM diff {
                              let CheckMe := 1;
                        }
                  }

            if (randomizer > prob) then
             repeat until (CheckMe = 1) {
                  let key := SortedMixTop[(Irand224() mod card(TOP))+1];
                        if (key in SurroundTOP) then {
                         fix Y[key] := 1;
                              let counter := counter + 1;
                              let SurroundTOP := SurroundTOP diff {key};
                              let CheckMe := 1;
                        }
                 }

if counter = W then break loop12;
```

```
}

        solve;
        if ((solve_result='solved' or solve_result_num=422)
and bestPromisingBound > cost) then {
         let Pro_OPEN := {};
            let bestPromisingBound := cost;
            for {j in J:Y[j]=1} let Pro_OPEN := Pro_OPEN union {j};
let mark := 3;
        }

      display Y;
      unfix Y;
let SurroundTOP := TOP;
let SurroundBOTTOM := BOTTOM;
}

let counter := 0;

if (SpecialKey in BOTTOM) then let copyBOTTOM := copyBOTTOM union {Specia
else let copyTOP := copyTOP union {SpecialKey};

if (mark = 0) then {
display 'mark0';

if NP < 9 then {
let NP := NP + 1;
for {j in Pro_OPEN} {
             if (j not in PUT_IN_ONE) then {
                      let PUT_IN_ONE := PUT_IN_ONE union {j};
if (j in copyBOTTOM) then let copyBOTTOM := copyBOTTOM diff {j};
else let copyTOP := copyTOP diff {j};
                         break;
                  }
         }
} else {
let NP := NPdefault;
let PUT_IN_ONE := {};
            let PUT_IN_ZERO := {};
let CheckMe := 0;
let copyTOP := TOP;
let copyBOTTOM := BOTTOM;

for {m in 1..10000} {
           if CheckMe = 0 then let randomizer := Uniform01();
```

```
        if (randomizer <= prob) then
                let key := SortedMixBottom[(Irand224() mod card(BOTTOM))+
                 let CheckMe := 1;
                 if (key in copyBOTTOM) then {
                        let PUT_IN_ONE := PUT_IN_ONE union {key};
                        let counter := counter + 1;
                        let copyBOTTOM := copyBOTTOM diff {key};
                        let CheckMe := 0;
                 }

           if (randomizer > prob) then
                 let key := SortedMixTop[(Irand224() mod card(TOP))+1];
                 let CheckMe := 1;
                 if (key in copyTOP) then {
                        let PUT_IN_ONE := PUT_IN_ONE union {key};
                        let counter := counter + 1;
                        let copyTOP := copyTOP diff {key};
                        let CheckMe := 0;
                }
           if counter = NP then break;
}
}
display PUT_IN_ONE;

}

if mark = 1 then {
display 'mark1';
      let PUT_IN_ONE := PUT_IN_ONE union {SpecialKey};
let copyTOP := TOP;
      let copyBOTTOM := BOTTOM;

      if NP < 9 then {
let NP := NP + 1;
for {j in PUT_IN_ONE} {
if (j in BOTTOM) then let copyBOTTOM := copyBOTTOM diff {j};
         else let copyTOP := copyTOP diff {j};
}
            for {j in PUT_IN_ZERO} {
             if (j in BOTTOM) then let copyBOTTOM := copyBOTTOM diff {j};
                  else let copyTOP := copyTOP diff {j};
            }
      } else {
       let NP := NPdefault;
         let PUT_IN_ONE := {};
```

```
        let PUT_IN_ZERO := {};
let counter := 0;

for {m in 1..10000} {
             if CheckMe = 0 then let randomizer := Uniform01();
                  if (randomizer <= prob) then {
                   let key := SortedMixBottom[(Irand224() mod card(BOTTOM
                        let CheckMe := 1;
                        if (key in copyBOTTOM) then {
                         let PUT_IN_ONE := PUT_IN_ONE union {key};
                              let counter := counter + 1;
                              let copyBOTTOM := copyBOTTOM diff {key};
                              let CheckMe := 0;
                        }
                  }

                  if (randomizer > prob) then {
                   let key := SortedMixTop[(Irand224() mod card(TOP))+1];
                        let CheckMe := 1;
                        if (key in copyTOP) then {
                         let PUT_IN_ONE := PUT_IN_ONE union {key};
                              let counter := counter + 1;
                              let copyTOP := copyTOP diff {key};
                              let CheckMe := 0;
                        }
                  }
                  if counter = NP then break;
             }
        }
        display PUT_IN_ONE;
}

if mark = 2 then {
display 'mark2';
let PUT_IN_ZERO := PUT_IN_ZERO union {SpecialKey};

display PUT_IN_ZERO;
display PUT_IN_ONE;

let copyTOP := TOP;
      let copyBOTTOM := BOTTOM;

      for {j in PUT_IN_ONE} {
       if (j in BOTTOM) then let copyBOTTOM := copyBOTTOM diff {j};
            else let copyTOP := copyTOP diff {j};
```

```
        }
        for {j in PUT_IN_ZERO} {
            if (j in BOTTOM) then let copyBOTTOM := copyBOTTOM diff {j};
           else let copyTOP := copyTOP diff {j};
        }
}

if mark = 3 then {
display 'backtrack';
let NP := NPdefault ;
let PUT_IN_ONE := {};
let PUT_IN_ZERO := {};

for {j in Pro_OPEN} {
let PUT_IN_ONE := PUT_IN_ONE union {j};
let counter := counter + 1;
if counter = NP then break;
}

display PUT_IN_ONE;
      let copyTOP := TOP;
      let copyBOTTOM := BOTTOM;

      for {j in PUT_IN_ONE} {
       if (j in BOTTOM) then let copyBOTTOM := copyBOTTOM diff {j};
            else let copyTOP := copyTOP diff {j};
      }
}
let CcopyTOP := copyTOP;
let CcopyBOTTOM := copyBOTTOM;

display bestPromisingBound;
display copyTOP;
display copyBOTTOM;
}

display bestPromisingBound;
display Pro_OPEN;
shell 'date';
}
```